Algoritmos e Estruturas de Dados

The Assignment Problem

	Job 1	Job 2	Job 3	Job 4
A	9	**2**	7	8
B	6	4	**3**	7
C	**5**	8	1	8
D	7	6	9	**4**

Data: 02-11-2019

Relatório realizado por:

Pedro Santos nMecº93221 33,3%

Pedro Amaral nMecº93283 33,3%

Diogo Cunha nMecº95278 33,3%

Índice

Análise de Resultados

1 Introdução

O Assignment Problem é um problema que consiste em encontrar a menor das somas de um gráfico bipartido de um determinado tamanho. Geralmente, é útil para atribuir um determinado número de tarefas a um determinado número de agentes, sendo que cada agente tem um custo diferente para cada tarefa.

Neste caso específico será considerado mesmo número de agentes e tarefas, ou seja, trata-se de um problema balanceado. No caso contrário em que os números de agentes e tarefas são diferentes trata-se de um caso não balanceado.

O objetivo é encontrar o custo total mínimo sendo que a cada agente é atribuída uma tarefa e todos os agentes têm tarefas diferentes.

2 Métodos Utilizados

Para chegar à solução utilizamos 3 métodos diferentes: brute force, brute force com branch and bound e permutações aleatórias.

- Brute Force:

  Neste método foram avaliadas todas as formas possíveis de atribuir n agentes a n tarefas.

  Ou seja, para cada permutação das tarefas calculamos o custo total e comparamos com os custos totais anteriores de forma a obter o custo total mínimo e máximo. Este método tem uma complexidade de O(n!).

- Brute Force com Branch and Bound:

  Este método é semelhante ao anterior, mas é mais rápido não avaliando várias permutações com chances bastante baixas de serem a melhor solução. Para isso o algoritmo recebe mais um argumento, o custo parcial, o qual é usado para decidir se deve ignorar determinadas permutações. Este método tem uma complexidade de O(2^n).

- Permutações Aleatórias:

  Este método gera um número elevado de permutações aleatórias (neste trabalho usamos 1 milhão) que são todas avaliadas. Este método nem sempre devolve a permutação ideal mas o custo mínimo que devolve é de grandeza semelhante ao custo mínimo ideal. Este método tem uma complexidade de O(n).

3 Mínimos e Máximos

3.1 Brute Force

		Seed					
		93221		**93283**		**95278**	
N	**N_Visited**	**Min_Cost**	**Max_Cost**	**Min_Cost**	**Max_Cost**	**Min_Cost**	**Max_Cost**
1	1	48	48	28	28	31	31
2	2	44	60	46	52	36	65
3	6	101	133	98	129	52	114
4	24	114	177	115	157	105	167
5	120	103	185	128	194	107	207
6	720	124	266	123	270	128	247
7	5040	154	315	149	305	141	301
8	40320	163	346	134	329	188	398
9	3,63E5	154	380	150	419	190	400
10	3,63E6	194	442	172	449	192	453
11	3,99E7	181	497	186	500	183	477
12	4,79E8	223	520	226	539	218	561
13	6,23E9	209	596	238	616	229	610
14	8,71E10	235	659	244	637	224	642

3.2 Random Permutations

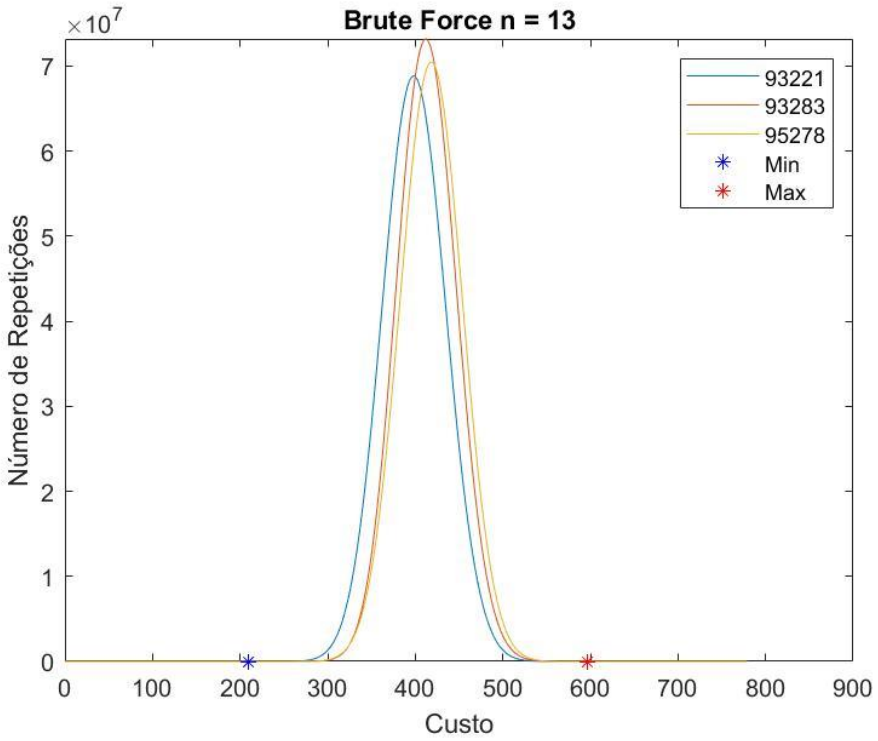
		Seed					
		93221		93283		95278	
N	N_Visited	Min_Cost	Max_Cost	Min_Cost	Max_Cost	Min_Cost	Max_Cost
1	1E6	48	48	28	28	31	31
2		44	60	46	52	36	65
3		101	133	98	129	52	114
4		114	177	115	157	105	167
5		103	185	128	194	107	207
6		124	266	123	270	128	247
7		154	315	149	305	141	301
8		163	346	134	329	188	398
9		154	380	150	419	190	400
10		197	442	173	449	196	447
11		206	484	195	484	198	469
12		241	508	245	520	233	537
13		237	579	269	576	254	593
14		276	607	295	612	254	597

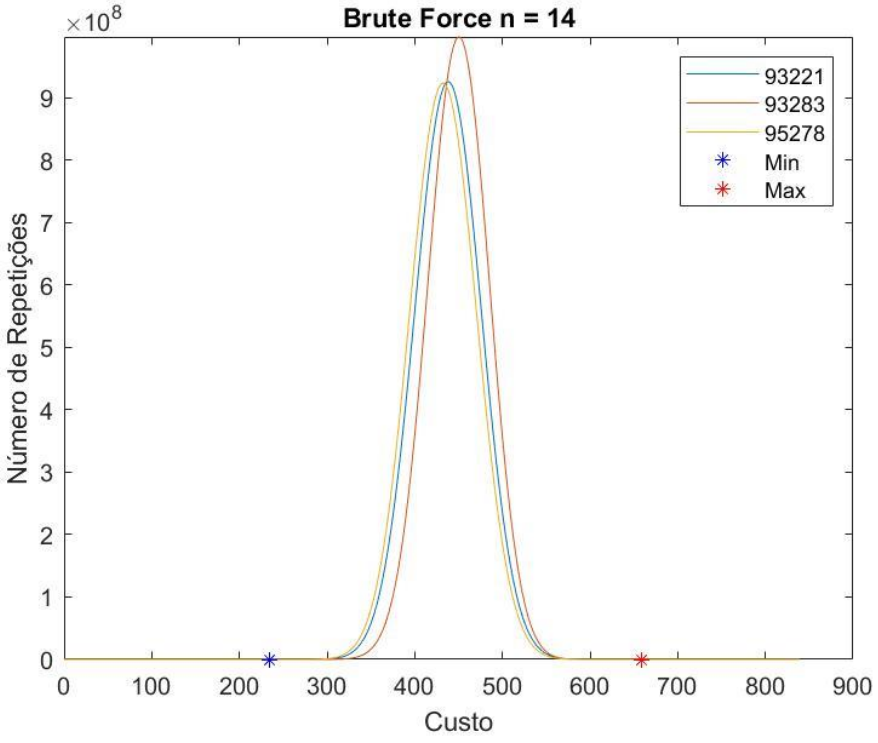
3.3 Branch and Bound

	Seed					
	93221		93283		95278	
N	N_Visited	Min_Cost	N_Visited	Min_Cost	N_Visited	Min_Cost
1	1	48	1	28	1	31
2	2	44	2	46	2	36
3	6	101	6	98	6	52
4	24	114	24	115	24	105
5	110	103	120	128	114	107
6	326	124	544	123	314	128
7	1874	154	1654	149	3156	141
8	5560	163	1702	134	10484	188
9	16732	154	6674	150	29652	190
10	20902	194	15008	172	74008	192
11	39338	181	55108	186	91658	183
12	2,76E5	223	9,67E5	226	3,33E5	218
13	9,06E5	209	1,96E6	238	4,72E5	229
14	2,33E6	235	1,66E6	244	1,42E6	224

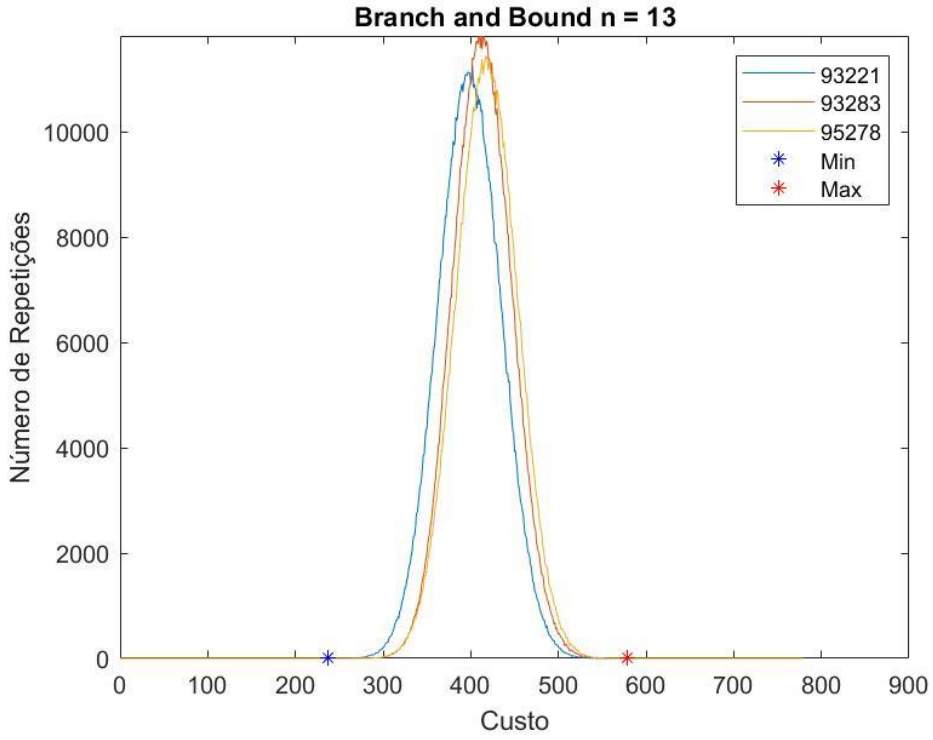
4 Número de Ocorrências dos Custos – Histogramas

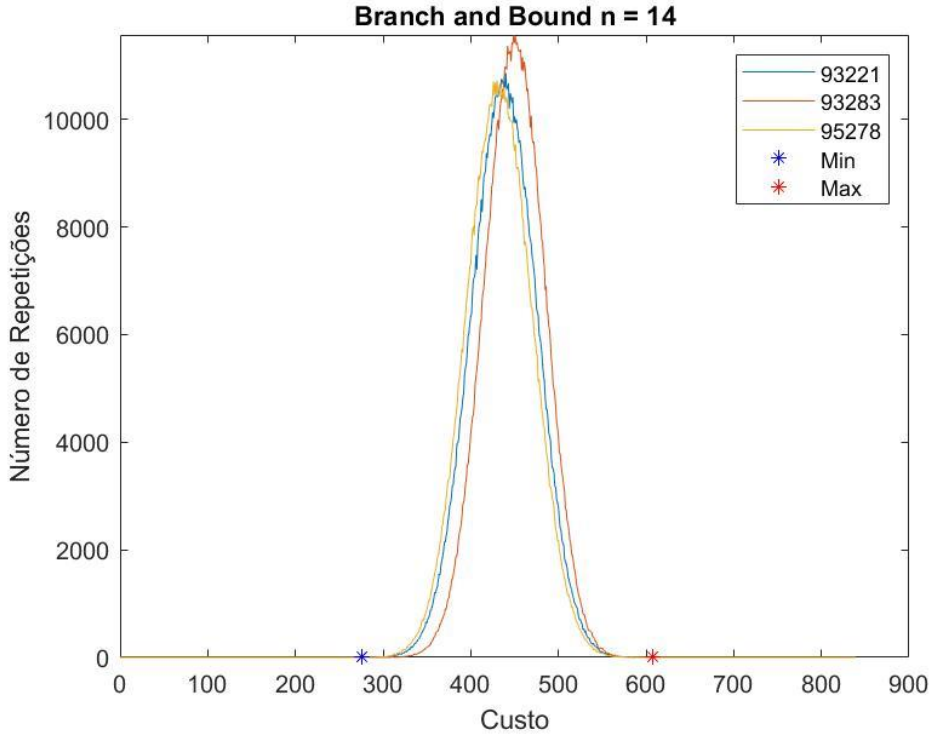
4.1 Brute Force



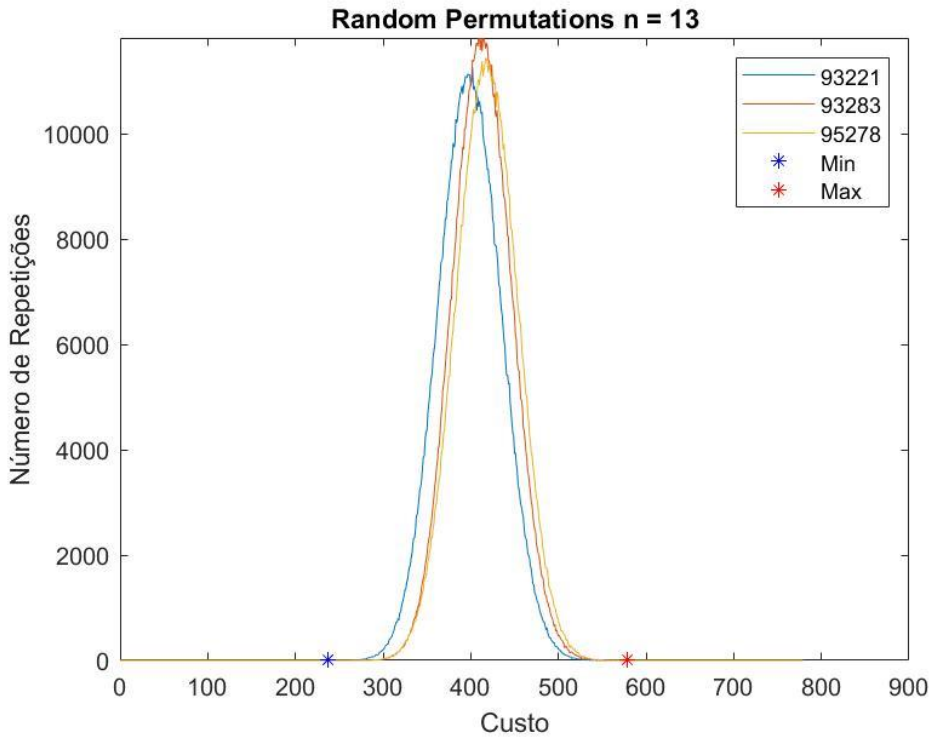


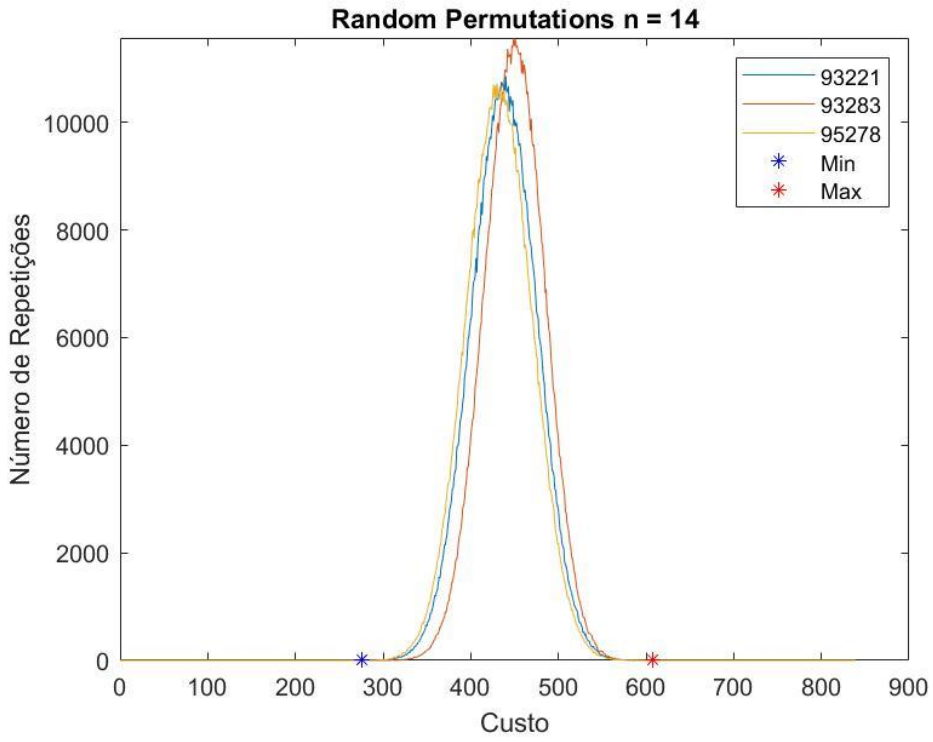
4.2 Branch and Bound





4.3 Random Permutations





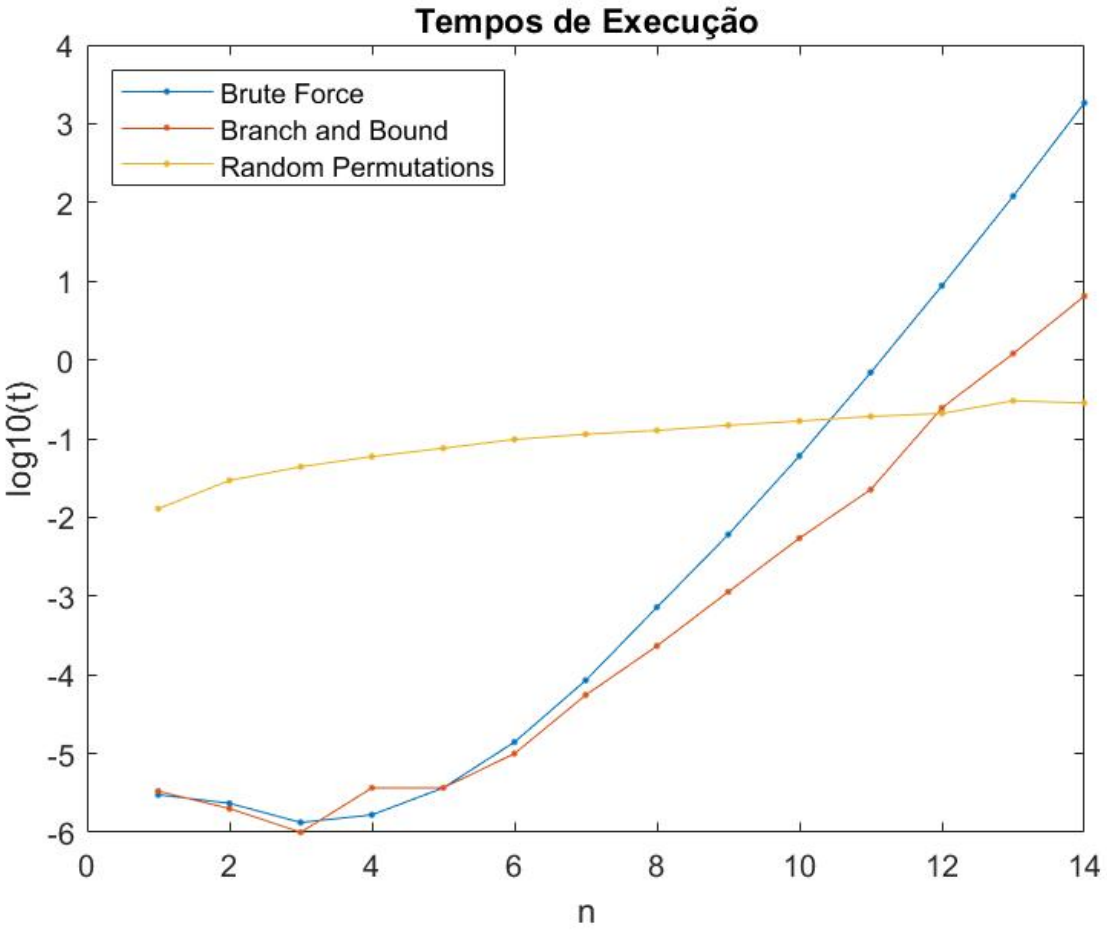
4.4 Análise de gráficos

Os gráficos dos histogramas de todos os métodos mostram uma distribuição aproximadamente gaussiana. De entre todos o método brute force apresenta uma gaussiana mais perfeita pois avalia todas as permutações possíveis. Os outros apresentam algumas irregularidades no gráfico devido ao facto de não avaliarem todas as permutações de tarefas.

N=13	Brute Force	Branch and Bound	Random Per.
Média - 93221	399	399	399
Média - 93221	413	413	413
média - 93283	418	418	418
Desvio padrão - 93283	36	36	36
Desvio padrão - 95278	34	34	34
Desvio padrão - 95278	35	35	35

N=14	Brute Force	Branch and Bound	Random Per.
Média - 93221	438	438	438
Média - 93221	450	450	450
Média - 93283	433	433	433
Desvio padrão - 93283	37	37	37
Desvio padrão - 95278	35	35	35
Desvio padrão - 95278	37	37	37

5 Tempos de Execução



Como podemos observar no gráfico dos tempos de execução, os gráficos do brute force e do branch and bound apresentam uma função linear quando usamos uma escala de y logaritmica. Isto acontece porque o número de permutações visitadas cresce exponencialmente à medida que o n máximo aumenta. Além disso, apartir de um determinado n, o tempo de execução do brute force é sempre superior ao tempo de branch and bound pois o branch and bound avalia muito menos permutações que o brute force. No caso, das random permutations observa-se que a taxa de crescimento do tempo de execução é notavelmente inferior, isto acontece porque são sempre avaliadas 1 milhão de permutações e apenas vai aumentando o número de elementos da permutação.

6 Bibliografia

- https://en.wikipedia.org/wiki/Assignment_problem
- https://www.youtube.com/watch?v=LDm4U9xYjwY&t=481s

7 Anexo (Código utilizado)

7.1 C

```
////////////////////////////////////////////////////////////////////////////////////////
//
// AED, 2019/2020
//
// 93221 - Pedro Santos
// 93283 - Pedro Amaral
// 95278 - Diogo Cunha
//
// Brute-force solution of the assignment problem (https://en.wikipedia.org/wiki/Assignment_problem)
//
// Compile with "cc -Wall -O2 assignment.c -lm" or equivalent
//
// In the assignment problem we will solve here we have n agents and n tasks; assigning agent
//   a
// to task
//   t
// costs
//   cost[a][t]
// The goal of the problem is to assign one agent to each task such that the total cost is minimized
// The total cost is the sum of the costs
//
// Things to do:
//   0. (mandatory)
//      Place the student numbers and names at the top of this file
//   1. (highly recommended)
//      Read and understand this code
//   2. (mandatory)
//      Modify the function generate_all_permutations to solve the assignment problem
//      Compute the best and worst solutions for all problems with sizes n=2,...,14 and for each
//      student number of the group
//   3. (mandatory)
//      Calculate and display an histogram of the number of occurrences of each cost
//      Does it follow approximately a normal distribution?
//      Note that the maximum possible cost is n * t_range
//   4. (optional)
//      For each problem size, and each student number of the group, generate one million (or more!)
//      random permutations and compute the best and worst solutions found in this way; compare
//      these solutions with the ones found in item 2
//      Compare the histogram computed in item 3 with the histogram computed using the random
//      permutations
//   5. (optional)
//      Try to improve the execution time of the program (use the branch-and-bound technique)
//   6. (optional)
//      Surprise us, by doing something more!
//   7. (mandatory)
//      Write a report explaining what you did and presenting your results
//


#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


//#define NDEBUG  // uncomment to skip disable asserts (makes the code slightly faster)
#include <assert.h>



////////////////////////////////////////////////////////////////////////////////////////
//
// problem data
//
// max_n ..... maximum problem size
// cost[a][t] ... cost of assigning agent a to task t
//


//
// if your compiler complains about srandom() and random(), replace #if 0 by #if 1
//
#if 0
# define srandom srand
# define random  rand
#endif


#define max_n    32          // do not change this (maximum number of agents, and tasks)
#define range    20          // do not change this (for the pseudo-random generation of costs)
#define t_range  (3 * range)  // do not change this (maximum cost of an assignment)
```

```
static int cost[max_n][max_n];
static int seed; // place a student number here!


static void init_costs(int n)
{
  if(n == -3)
  { // special case (example for n=3)
    cost[0][0] = 3; cost[0][1] = 8; cost[0][2] = 6;
    cost[1][0] = 4; cost[1][1] = 7; cost[1][2] = 5;
    cost[2][0] = 5; cost[2][1] = 7; cost[2][2] = 5;
    return;
  }
  if(n == -5)
  { // special case (example for n=5)
    cost[0][0] = 27; cost[0][1] = 27; cost[0][2] = 25; cost[0][3] = 41; cost[0][4] = 24;
    cost[1][0] = 28; cost[1][1] = 26; cost[1][2] = 47; cost[1][3] = 38; cost[1][4] = 21;
    cost[2][0] = 22; cost[2][1] = 48; cost[2][2] = 26; cost[2][3] = 14; cost[2][4] = 24;
    cost[3][0] = 32; cost[3][1] = 31; cost[3][2] =  9; cost[3][3] = 41; cost[3][4] = 36;
    cost[4][0] = 24; cost[4][1] = 34; cost[4][2] = 30; cost[4][3] = 35; cost[4][4] = 45;
    return;
  }
  assert(n >= 1 && n <= max_n);
  srandom((unsigned int)seed * (unsigned int)max_n + (unsigned int)n);
  for(int a = 0;a < n;a++)
    for(int t = 0;t < n;t++)
      cost[a][t] = 3 + (random() % range) + (random() % range) + (random() % range); // [3,3*range]
}




////////////////////////////////////////////////////////////////////////////////
//
// code to measure the elapsed time used by a program fragment (an almost copy of elapsed_time.h)
//
// use as follows:
//
//   (void)elapsed_time();
//   // put your code to be time measured here
//   dt = elapsed_time();
//   // put morecode to be time measured here
//   dt = elapsed_time();
//
// elapsed_time() measures the CPU time between consecutive calls
//


#if defined(__linux__) || defined(__APPLE__)


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


#include <time.h>


static 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)
    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>


static 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




////////////////////////////////////////////////////////////////////////////////
//
// function to generate a pseudo-random permutation
//


void random_permutation(int n,int t[n])
{
  assert(n >= 1 && n <= 1000000);
```

```
  for(int i = 0;i < n;i++)
    t[i] = i;
  for(int i = n - 1;i > 0;i--)
  {
    int j = (int)floor((double)(i + 1) * (double)random() / (1.0 + (double)RAND_MAX)); // range 0..i
    assert(j >= 0 && j <= i);
    int k = t[i];
    t[i] = t[j];
    t[j] = k;
  }
}



////////////////////////////////////////////////////////////////////////////////////////////
//
// place to store best and worst solutions (also code to print them)
//


static int min_cost,min_cost_assignment[max_n]; // smallest cost information
static int max_cost,max_cost_assignment[max_n]; // largest cost information
static long n_visited; // number of permutations visited (examined)
static int histogram[14*t_range]; // histogram global variable
static double cpu_time;


#define minus_inf  -1000000000  // a very small integer
#define plus_inf   +1000000000  // a very large integer


static void reset_solutions(void)
{
  min_cost = plus_inf;
  max_cost = minus_inf;
  n_visited = 0l;
  // place your histogram initialization code here
  memset(histogram, 0, sizeof(histogram));
  cpu_time = 0.0;
}


#define show_info_1        (1 << 0)
#define show_info_2        (1 << 1)
#define show_costs         (1 << 2)
#define show_min_solution  (1 << 3)
#define show_max_solution  (1 << 4)
#define show_histogram     (1 << 5)
#define show_all           (0xFFFF)


static void show_solutions(int n,char *header,int what_to_show)
{
  printf("%s\n",header);
  if((what_to_show & show_info_1) != 0)
  {
    printf("  seed ......... %d\n",seed);
    printf("  n ............. %d\n",n);
  }
  if((what_to_show & show_info_2) != 0)
  {
    printf("  visited ....... %ld\n",n_visited);
    printf("  cpu time ...... %.3fs\n",cpu_time);
  }
  if((what_to_show & show_costs) != 0)
  {
    printf("  costs .........");
    for(int a = 0;a < n;a++)
    {
      for(int t = 0;t < n;t++)
        printf(" %2d",cost[a][t]);
      printf("\n%s",(a < n - 1) ? "                " : "");
    }
  }
  if((what_to_show & show_min_solution) != 0)
  {
    printf("  min cost ...... %d\n",min_cost);
    if(min_cost != plus_inf)
    {
      printf("  assignment ...");
      for(int i = 0;i < n;i++)
        printf(" %d",min_cost_assignment[i]);
      printf("\n");
    }
  }
  if((what_to_show & show_max_solution) != 0)
  {
    printf("  max cost ...... %d\n",max_cost);
    if(max_cost != minus_inf)
    {
      printf("  assignment ...");
      for(int i = 0;i < n;i++)
        printf(" %d",max_cost_assignment[i]);
      printf("\n");
    }
  }
  if((what_to_show & show_histogram) != 0)
  {
    // place your code to print the histogram here


                        printf("\nHistogram n = %d\n", n);
                        printf("i\tn[i]\n");
    for (int i = 0; i < n*t_range; i++)
    {
                                printf("%d\t%d\n", i, histogram[i]);
                        }
  }
}



////////////////////////////////////////////////////////////////////////////////////////////
//
// code used to generate all permutations of n objects (brute force)
//
// n ........ number of objects
// m ........ index where changes occur (a[0], ..., a[m-1] will not be changed)
// a[idx] ... the number of the object placed in position idx
//
```

```
static void generate_all_permutations(int n,int m,int a[n])
{
  if(m < n - 1)
  {
    //
    // not yet at the end; try all possibilities for a[m]
    //
    for(int i = m;i < n;i++)
    {
#define swap(i,j)  do { int t = a[i]; a[i] = a[j]; a[j] = t; } while(0)
      swap(i,m);                          // exchange a[i] with a[m]
      generate_all_permutations(n,m + 1,a); // recurse
      swap(i,m);                          // undo the exchange of a[i] with a[m]
#undef swap
    }
  }
  else
  {
    //
    // visit the permutation
    //
    int totalcost=0;
    for (int j=0; j<n; j++) {
        totalcost += cost[j][a[j]];
    }


    n_visited++;


    // place your code to update the best and worst solutions, and to update the histogram here
    if (max_cost < totalcost) {max_cost = totalcost; memcpy(max_cost_assignment, a, sizeof(max_cost_assignment));}
    if (min_cost > totalcost) {min_cost = totalcost; memcpy(min_cost_assignment, a, sizeof(min_cost_assignment));}
                        histogram[totalcost]++;


  }
}


////////////////////////////////////////////////////////////////////////////////////////////
//
// code used to solve the problem with branch and bound
//
// n ..... number of objects
// m ..... index where changes occur (a[0], ..., a[m-1] will not be changed)
// a[idx] ... the number of the object placed in position idx
// custoparcial
//


static void generate_all_permutations_branch_and_bound(int n,int m,int a[n], int custoparcial)
{


    if(m < n - 1)
    {
        for(int i = m;i < n;i++)
        {
            if (custoparcial > min_cost) break;
#define swap(i,j)  do { int t = a[i]; a[i] = a[j]; a[j] = t; } while(0)
            swap(i,m);                          // exchange a[i] with a[m]
            generate_all_permutations_branch_and_bound(n,m + 1,a, custoparcial + cost[m][a[m]]); // recurse
            swap(i,m);                          // undo the exchange of a[i] with a[m]
#undef swap
        }
    }
    else
    {
    //
    // visit the permutation and update the best solution
    //
    if (custoparcial + cost[m][a[m]] < min_cost) {
        min_cost = custoparcial + cost[m][a[m]];
        memcpy(min_cost_assignment, a, sizeof(min_cost_assignment));
    }


    n_visited++;
    histogram[custoparcial + cost[m][a[m]]]++;


  }


}




////////////////////////////////////////////////////////////////////////////////////////////


//code to solve  the problem with random permutations
//
// n ..... number of objects
// t[n] ..... permutation


void calculate_solutions(int n, int t[n])
{
                        for (int i = 0; i < 1000000; i++)
                        {
                                                random_permutation(n, t);
                                                int totalcost=0;
                                                for (int j=0; j<n; j++) {
                                                                        totalcost += cost[j][t[j]];
                                                }


                                                n_visited++;


                                                if (max_cost < totalcost) {max_cost = totalcost; memcpy(max_cost_assignment, t, sizeof(max_cost_assignment));}
                                                if (min_cost > totalcost) {min_cost = totalcost; memcpy(min_cost_assignment, t, sizeof(min_cost_assignment));}
                                                histogram[totalcost]++;
                        }
```

```
}

////////////////////////////////////////////////////////////////////////////////////////////////////


//code to print the data to a file and use in matlab
//
// n - number of objects
// o - 1->brute force, 2->branch and bound, 3->random permutations


void print_to_file(int n, int o)
{
    FILE *fp;
    switch (o)
    {
                                case 1:  //brute force
        // histogram
        if (n==13) {
            fp = fopen("histogram13.txt", "w");
            for (int i = 0; i < n*t_range; i++)
            {
                fprintf(fp, "%d\t%d\n", i, histogram[i]);
            }
            fclose(fp);
        }
        if (n==14) {
            fp = fopen("histogram14.txt", "w");
            for (int i = 0; i < n*t_range; i++)
            {
                fprintf(fp, "%d\t%d\n", i, histogram[i]);
            }
            fclose(fp);
        }
        // tempos de execução
                                        fp = fopen("temposbruteforce.txt", "a");
        fprintf(fp, "%d\t%f\n", n, cpu_time);
        fclose(fp);
        break;


                                case 2:
                                        fp = fopen("temposbranchandbound.txt", "a");
        fprintf(fp, "%d\t%f\n", n, cpu_time);
        fclose(fp);
        break;
    case 3:
        fp = fopen("tempospermutacoesaleatorias.txt", "a");
        fprintf(fp, "%d\t%f\n", n, cpu_time);
        fclose(fp);
        break;


    case 4:
                                        if (n==13) {
                                                fp = fopen("randomHistogram13.txt", "w");
                                                for (int i = 0; i < n*t_range; i++)
                                                {
                                                        fprintf(fp, "%d\t%d\n", i, histogram[i]);
                                                }
                                                fclose(fp);
                                        }
        if (n==14) {
            fp = fopen("randomHistogram14.txt", "w");
            for (int i = 0; i < n*t_range; i++)
            {
                fprintf(fp, "%d\t%d\n", i, histogram[i]);
            }
            fclose(fp);
        }
    case 5:
                                        if (n==13) {
                                                fp = fopen("BaBHistogram13.txt", "w");
                                                for (int i = 0; i < n*t_range; i++)
                                                {
                                                        fprintf(fp, "%d\t%d\n", i, histogram[i]);
                                                }
                                                fclose(fp);
                                        }
        if (n==14) {
            fp = fopen("BaBHistogram14.txt", "w");
            for (int i = 0; i < n*t_range; i++)
            {
                fprintf(fp, "%d\t%d\n", i, histogram[i]);
            }
            fclose(fp);
        }
                                case 6:
                                        fp = fopen("minimosMaximosNVisitasBF.txt", "a");
                                        fprintf(fp, "Para n = %d\n", n );
                                        fprintf(fp, "Min: %d\n", min_cost);
                                        fprintf(fp, "Max: %d\n", max_cost);
                                        fprintf(fp, "n_visited: %ld\n", n_visited);
                                        fclose(fp);
                                        break;
                                case 7:
                                        fp = fopen("minimosMaximosNVisitasRP.txt", "a");
                                        fprintf(fp, "Para n = %d\n", n );
                                        fprintf(fp, "Min: %d\n", min_cost);
                                        fprintf(fp, "Max: %d\n", max_cost);
                                        fprintf(fp, "n_visited: %ld\n", n_visited);
                                        fclose(fp);
                                        break;
                                case 8:
                                        fp = fopen("minimosMaximosNVisitasBaB.txt", "a");
                                        fprintf(fp, "Para n = %d\n", n );
                                        fprintf(fp, "Min: %d\n", min_cost);
                                        fprintf(fp, "Max: %d\n", max_cost);
                                        fprintf(fp, "n_visited: %ld\n", n_visited);
                                        fclose(fp);
                                        break;
                }
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//
// main program
//
```

```
int main(int argc,char **argv)
{
  if(argc == 2 && argv[1][0] == '-' && argv[1][1] == 'e')
  {
    seed = 0;
    {
      int n = 3;
      init_costs(-3); // costs for the example with n = 3
      int a[n];
      for(int i = 0;i < n;i++)
        a[i] = i;
      reset_solutions();
      (void)elapsed_time();
      generate_all_permutations(n,0,a);
      cpu_time = elapsed_time();
      show_solutions(n,"Example for n=3",show_all);
      printf("\n");
    }
    {
      int n = 5;
      init_costs(-5); // costs for the example with n = 5
      int a[n];
      for(int i = 0;i < n;i++)
        a[i] = i;
      reset_solutions();
      (void)elapsed_time();
      generate_all_permutations(n,0,a);
      cpu_time = elapsed_time();
      show_solutions(n,"Example for n=5",show_all);
      return 0;
    }
  }
  if(argc == 2)
  {
    seed = atoi(argv[1]); // seed = student number
    if(seed >= 0 && seed <= 1000000)
    {
      for(int n = 1;n <= max_n;n++)
      {
        init_costs(n);
        show_solutions(n,"Problem statement",show_info_1 | show_costs);
        //
        // 2.
        //
        if(n <= 14) // use a smaller limit here while developing your code(14)
        {
          int a[n];
          for(int i = 0;i < n;i++)
            a[i] = i; // initial permutation
          reset_solutions();
          (void)elapsed_time();
          generate_all_permutations(n,0,a);
          cpu_time = elapsed_time();
          show_solutions(n,"Brute force",show_info_2 | show_min_solution | show_max_solution | show_histogram);
          print_to_file(n, 1);
          print_to_file(n, 6);
        }
        //
        // place here your code that solves the problem with branch-and-bound
        //
        //less_row_cost = +1000000000;


        if(n <= 14) // use a smaller limit here while developing your code(16)
        {
          int a[n];
          for(int i = 0;i < n;i++)
            a[i] = i; // initial permutation
          reset_solutions();
          (void)elapsed_time();
          generate_all_permutations_branch_and_bound(n,0,a,0);
          cpu_time = elapsed_time();
          show_solutions(n,"Brute force with branch-and-bound",show_info_2 | show_min_solution);
          print_to_file(n, 2);
          print_to_file(n, 5);
          print_to_file(n, 8);
        }


        //
        // place here your code that generates the random permutations
        //


                                        if (n <= 14)
                                        {
                                                int t[n];
                                                reset_solutions();
                                                (void)elapsed_time();
                                                calculate_solutions(n, t);
                                                cpu_time = elapsed_time();
                                                show_solutions(n,"Random Permutation",show_info_2 | show_min_solution | show_max_solution);
                                                print_to_file(n, 3);
                                                print_to_file(n, 4);
                                                print_to_file(n, 7);
                                        }


        //
        // done
        //
        printf("\n");
      }
      return 0;
    }
  }
  fprintf(stderr,"usage: %s -e             # for the examples\n",argv[0]);
  fprintf(stderr,"usage: %s student_number\n",argv[0]);
  return 1;
}
```

7.2 Matlab

Histogram:

```
formatSpec = '%d %d'; % Ler 2 inteiros
sizeA = [2 Inf]; % Inf = Ler até ao final do ficheiro

% seed = 93221
% Ler quando n = 13
fileID = fopen("93221/randomHistogram13.txt",'r');
results1393221 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);
% Ler quando n = 14
fileID = fopen('93221/randomHistogram14.txt','r');
results1493221 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);

% seed = 93283
% Ler quando n = 13
fileID = fopen("93283/randomHistogram13.txt",'r');
results1393283 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);
% Ler quando n = 14
fileID = fopen('93283/randomHistogram14.txt','r');
results1493283 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);

% seed = 95278
% Ler quando n = 13
fileID = fopen("95278/randomHistogram13.txt",'r');
results1395278 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);
% Ler quando n = 14
fileID = fopen('95278/randomHistogram14.txt','r');
results1495278 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);

% Histograma n = 13:
x93221 = results1393221(:,1);
y93221 = results1393221(:,2);
x93283 = results1393283(:,1);
y93283 = results1393283(:,2);
x95278 = results1395278(:,1);
y95278 = results1395278(:,2);

plot(x93221, y93221, x93283, y93283, x95278, y95278);
ylabel('Número de Repetições');
xlabel('Custo');
title('Random Permutations n = 13');
axis([0 900 0 inf]);
hold on
aux = x93221(y93221>0);
xmin = min(aux);
ymin = y93221(xmin + 1);
xmax = max(aux);
ymax = y93221(xmax + 1);
plot(xmin, ymin, 'b*')
plot(xmax, ymax, 'r*')
legend('93221', '93283', '95278', 'Min', 'Max');

% Histograma n = 14:
figure(2);
x93221 = results1493221(:,1);
y93221 = results1493221(:,2);
x93283 = results1493283(:,1);
y93283 = results1493283(:,2);
x95278 = results1495278(:,1);
y95278 = results1495278(:,2);

plot(x93221, y93221, x93283, y93283, x95278, y95278);
legend('93221', '93283', '95278');
ylabel('Número de Repetições');
xlabel('Custo');
title('Random Permutations n = 14');
axis([0 900 0 inf]);
hold on
aux = x93221(y93221>0);
xmin = min(aux);
ymin = y93221(xmin + 1);
xmax = max(aux);
ymax = y93221(xmax + 1);
plot(xmin, ymin, 'b*')
plot(xmax, ymax, 'r*')
legend('93221', '93283', '95278', 'Min', 'Max');
```

Tempos de Execução:

```
clear;
clc;

formatSpec = '%d %f'; % Ler 2 inteiros
sizeA = [2 Inf]; % Inf = Ler até ao final do ficheiro

% ---------- Brute Force ----------
% seed = 93221
fileID = fopen("93221/temposbruteforce.txt",'r');
results1393221 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);
% seed = 93283
fileID = fopen("93283/temposbruteforce.txt",'r');
results1393283 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);
% seed = 95278
fileID = fopen("95278/temposbruteforce.txt",'r');
results1395278 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);

% Histograma n = 13:
x93221 = results1393221(:,1);
y93221 = results1393221(:,2);
x93283 = results1393283(:,1);
y93283 = results1393283(:,2);
x95278 = results1395278(:,1);
y95278 = results1395278(:,2);
YBF = zeros(14, 3);
X = [1:14]';
for k = 1:14
                    YBF(k, 1) = results1393221(k,2);
                    YBF(k, 2) = results1393283(k,2);
                    YBF(k, 3) = results1395278(k,2);
```

```
end
YBF = sum(YBF, 2);
YBF = YBF./3;
%plot(XBF, log10(YBF), '.-');

% ------------------- Branch and Bound --------------------
% seed = 93221
fileID = fopen("93221/temposbranchandbound.txt",'r');
results1393221 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);
% seed = 93283
fileID = fopen("93283/temposbranchandbound.txt",'r');
results1393283 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);
% seed = 95278
fileID = fopen("95278/temposbranchandbound.txt",'r');
results1395278 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);

% Histograma n = 13:
x93221 = results1393221(:,1);
y93221 = results1393221(:,2);
x93283 = results1393283(:,1);
y93283 = results1393283(:,2);
x95278 = results1395278(:,1);
y95278 = results1395278(:,2);
YBaB = zeros(14, 3);

for k = 1:14
                    YBaB(k, 1) = results1393221(k,2);
                    YBaB(k, 2) = results1393283(k,2);
                    YBaB(k, 3) = results1395278(k,2);
end

YBaB = sum(YBaB, 2);
YBaB = YBaB./3;

% ------------------- Random Permutations --------------------
% seed = 93221
fileID = fopen("93221/tempospermutacoesaleatorias.txt",'r');
results1393221 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);
% seed = 93283
fileID = fopen("93283/tempospermutacoesaleatorias.txt",'r');
results1393283 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);
% seed = 95278
fileID = fopen("95278/tempospermutacoesaleatorias.txt",'r');
results1395278 = fscanf(fileID,formatSpec,sizeA)';
fclose(fileID);

% Histograma n = 13:
x93221 = results1393221(:,1);
y93221 = results1393221(:,2);
x93283 = results1393283(:,1);
y93283 = results1393283(:,2);
x95278 = results1395278(:,1);
y95278 = results1395278(:,2);
YRP = zeros(14, 3);

for k = 1:14
                    YRP(k, 1) = results1393221(k,2);
                    YRP(k, 2) = results1393283(k,2);
                    YRP(k, 3) = results1395278(k,2);
end

YRP = sum(YRP, 2);
YRP = YRP./3;

plot(X, log10(YBF), '.-', X, log10(YBaB), '.-',  X, log10(YRP), '.-');

legend('Brute Force', 'Branch and Bound', 'Random Permutations', 'location', 'northwest');
ylabel('log10(t)');
xlabel('n');
title('Tempos de Execução');
```