



Nomes: Josué Nunes Campos – 03465
Lucas Barros Pereira Costa – 03511
Mateus Coelho Santos – 03488

RELATÓRIO FINAL

- **Objetivo**

O programa implementado tem como objetivo calcular a melhor distância possível de um certo número N de entradas através de permutações sucessivas, avaliando todas as possíveis caminhos entre as cidades.

Desenvolvemos este programa com o intuito de obter o tempo gasto para um número crescente de N, começando de 2 e incrementando 1 até 11, já que para N maior que 11 o tempo necessário seria muito extenso (com N=12, por exemplo, o nosso programa estava demorando mais de 6 horas).

- **Da Implementação**

Começamos o processo de desenvolvimento procurando um algoritmo de permutação capaz de gerar todos os possíveis arranjos de cidades. Após uma pesquisa na internet escolhemos o seguinte algoritmo:

```
4  #include <stdio.h>
5
6  void troca(int vetor[], int i, int j)
7  {
8      int aux = vetor[i];
9      vetor[i] = vetor[j];
10     vetor[j] = aux;
11 }
12
13 void permuta(int vetor[], int inf, int sup)
14 {
15     int i;
16     if(inf == sup)
17     {
18         for(i = 0; i <= sup; i++)
19             printf("%d ", vetor[i]);
20         printf("\n");
21     }
22     else
23     {
24         for(i = inf; i <= sup; i++)
25         {
26             troca(vetor, inf, i);
27             permuta(vetor, inf + 1, sup);
28             troca(vetor, inf, i); // backtracking
29         }
30     }
31 }
32
33 int main(int argc, char *argv[])
34 {
35     int v[] = {1, 2, 3, 4};
36     int tam_v = sizeof(v) / sizeof(int);
37
38     permuta(v, 0, tam_v - 1);
39
40     return 0;
41 }
```

<https://gist.github.com/marcoscastro/60f8f82298212e267021>

Este algoritmo funciona de forma recursiva, sendo que a cada chamada do subprograma ele realiza a troca de posição das cidades, incrementando 1 no limite inferior, de forma que a condição de parada seja quando o limite inferior for igual ao superior. Após atingir a condição de parada, ele exibe cada permutação feita e volta a troca para o arranjo anterior e a exibe na tela, repetindo esse processo até chegar na chamada inicial do subprograma.

```
for(i = inf; i <= sup; i++)  
{  
    troca(vetor, inf, i);  
    permuta(vetor, inf + 1, sup);  
    troca(vetor, inf, i); // backtracking  
}
```

Com a escolha do algoritmo feita, optamos por modificar o subprograma “permuta”, de forma que pudéssemos inserir a matriz de distância e realizar o cálculo da distância de cada permutação. Além disso, colocamos a execução do cálculo da menor distância possível nessa modificação, sendo possível guardar o melhor caminho também, e ao final da execução desse subprograma, obtém-se todas as informações necessárias para finalizar o programa.

Com essas alterações feitas e os problemas resolvidos, o algoritmo ficou da seguinte forma:

```

38 void permuta(int X, int *Vetor, int inf, int sup, int **Matriz, int *Melhor_Cam, int *Menor_Dist, int *cont){
39     if(inf == sup){
40         int Distancia, i;
41         Distancia = Matriz[X][Vetor[1]];
42         Distancia += Matriz[Vetor[sup]][X];
43         printf("%d ", X);
44         for(i = 1; i <= sup; i++){
45             printf("%d ", Vetor[i]);
46             if(i != sup){
47                 Distancia += Matriz[Vetor[i]][Vetor[i+1]];
48             }
49         }
50         printf("%d - Distancia: %d\n", X, Distancia);
51         printf("\n");
52         if(*cont == 0){
53             *Menor_Dist = Distancia;
54             for(i=1; i<= sup; i++){
55                 Melhor_Cam[i] = Vetor[i];
56             }
57             *cont = 1;
58         }
59         else{
60             if(Distancia < *Menor_Dist){
61                 *Menor_Dist = Distancia;
62                 for(i=1; i<= sup; i++){
63                     Melhor_Cam[i] = Vetor[i];
64                 }
65             }
66         }
67     }
68     else{
69         int i;
70         for(i = inf; i <= sup; i++){
71             troca(Vetor, inf, i);
72             permuta(X, Vetor, inf+1, sup, Matriz, Melhor_Cam, Menor_Dist, cont);
73             troca(Vetor, inf, i);
74         }
75     }
76 }

```

- **Dos Resultados Obtidos**

Desenvolvido o programa, partimos para a análise das entradas, começando com N=2 e incrementando 1 até chegarmos em N=11. Consideramos a matrícula de cada integrante do grupo (3465, 3488 e 3511) para todos os testes e as distâncias entre as cidades geradas aleatoriamente pela função disponível da linguagem C chamada “rand”.

Obtivemos o tempo de cada entrada exibido na tabela abaixo, sendo que as configurações da máquina sujeita aos testes é:

Processador: Intel ® Core™ i3 - 4005U CPU @ 1.70GHz;

Memória RAM: 4 GB;

HD: 500 GB;

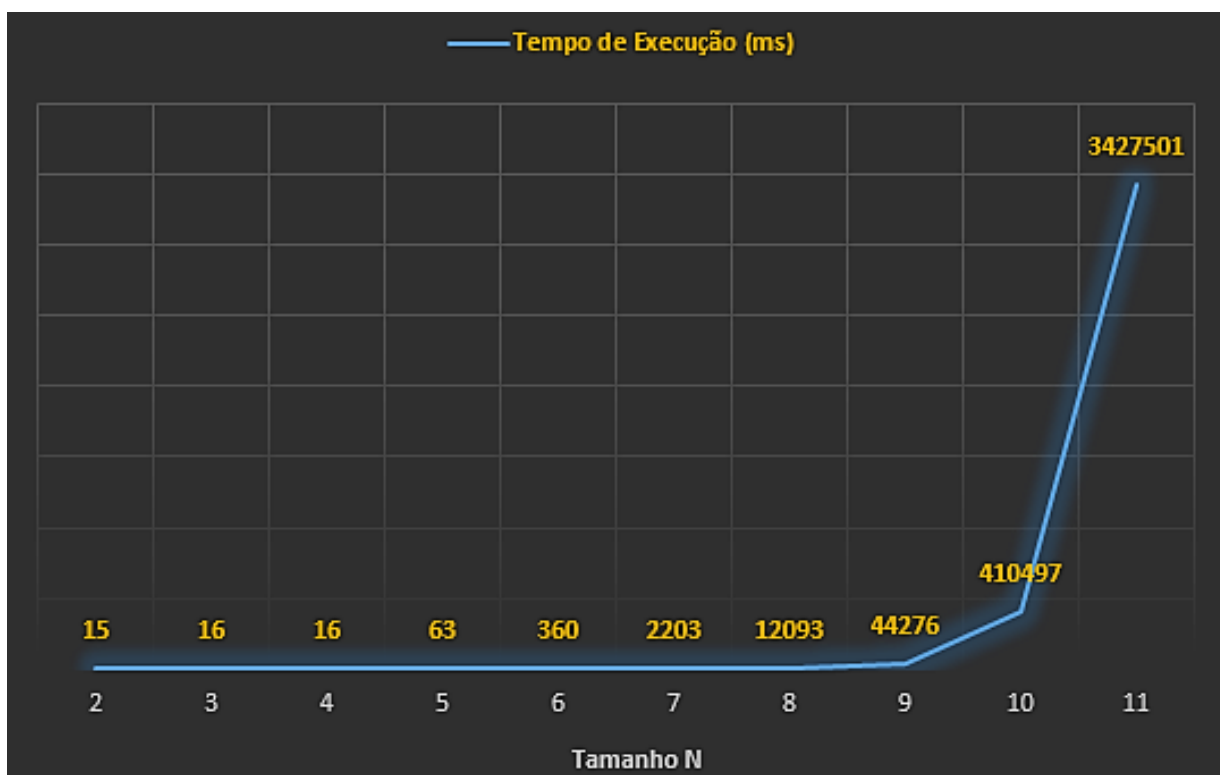
Sistema Operacional: Windows 10 Home;

IDE utilizada na implementação: Dev C++ 5.11;

Compilador: MinGW, GCC no Windows.

N \ Tempo	Milissegundos	Segundos	Minutos	Horas
2	15,00 ms.	0,01 s.	0,00 min.	0,00 hrs.
3	16, 00 ms.	0,02 s.	0,00 min.	0,00 hrs.
4	16, 00 ms.	0,02 s.	0,00 min.	0,00 hrs.
5	63,00 ms.	0,06 s.	0,00 min.	0,00 hrs.
6	360,00 ms.	0,36 s.	0,01 min.	0,00 hrs.
7	2203,00 ms.	2,20 s.	0,04 min.	0,00 hrs.
8	12093,00 ms.	12,09 s.	0,20 min.	0,00 hrs.
9	44276,00 ms.	44,28 s.	0,74 min.	0,01 hrs.
10	410497,00 ms.	410,30 s.	6,84 min.	0,11 hrs.
11	3427501,00 ms.	3427,50 s.	57,13 min.	0,95 hrs.

Utilizando os tempos de execução adquiridos acima, pudemos gerar o gráfico (N x Tempo de Execução) abaixo:



- **Conclusão**

Diante dos resultados obtidos, pudemos concluir que, a análise de complexidade de um algoritmo é fundamental para determinar a usabilidade do mesmo, visto que, para um número grande de cidades, não é possível obter um resultado rapidamente. Vimos também que, para esse trabalho, a partir do aumento de cidades o tempo de execução muda drasticamente, evidenciando como a complexidade fatorial atua na prática.

Durante a execução do trabalho, discutimos o caso da viabilidade desse programa caso fossemos contratados por uma empresa para calcularmos a melhor rota possível para seus caminhões. Chegamos à conclusão de que não seria possível usar esse programa para tal problema, justamente pelo fato de haver a possibilidade de uma rota possuir muitas cidades, logo, o resultado não seria obtido de imediato e teríamos problemas em dar um resultado para a empresa.

Portanto, como solução, buscaríamos outra forma mais eficiente de para resolver o problema da empresa, levando em consideração o que foi aprendido em sala de aula, de que pode haver certos problemas que não possuem maneiras de os tornarem mais eficientes. Ademais, o trabalho garantiu para todos do grupo, conhecimento acerca de como avaliar a complexidade de algoritmos e como essa complexidade pode influenciar nas tomadas de decisão para resolver um determinado problema da vida real.