Universidade Federal de Viçosa - Campus Florestal
CCF251 – Algoritmos e Estrutura de Dados I
2018/2
Profa. Thais R. M. Braga Silva

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 os 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).

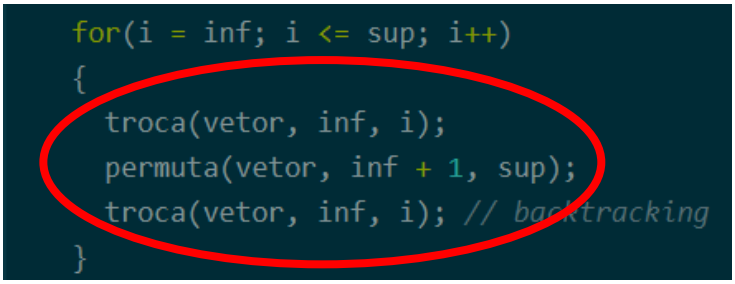
- **Da Implementação**

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:

```c
#include <stdio.h>

void troca(int vetor[], int i, int j)
{
  int aux = vetor[i];
  vetor[i] = vetor[j];
  vetor[j] = aux;
}

void permuta(int vetor[], int inf, int sup)
{
  int i;
  if(inf == sup)
  {
    for(i = 0; i <= sup; i++)
      printf("%d ", vetor[i]);
    printf("\n");
  }
  else
  {
    for(i = inf; i <= sup; i++)
    {
      troca(vetor, inf, i);
      permuta(vetor, inf + 1, sup);
      troca(vetor, inf, i); // backtracking
    }
  }
}

int main(int argc, char *argv[])
{
  int v[] = {1, 2, 3, 4};
  int tam_v = sizeof(v) / sizeof(int);

  permuta(v, 0, tam_v - 1);

  return 0;
}
```

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

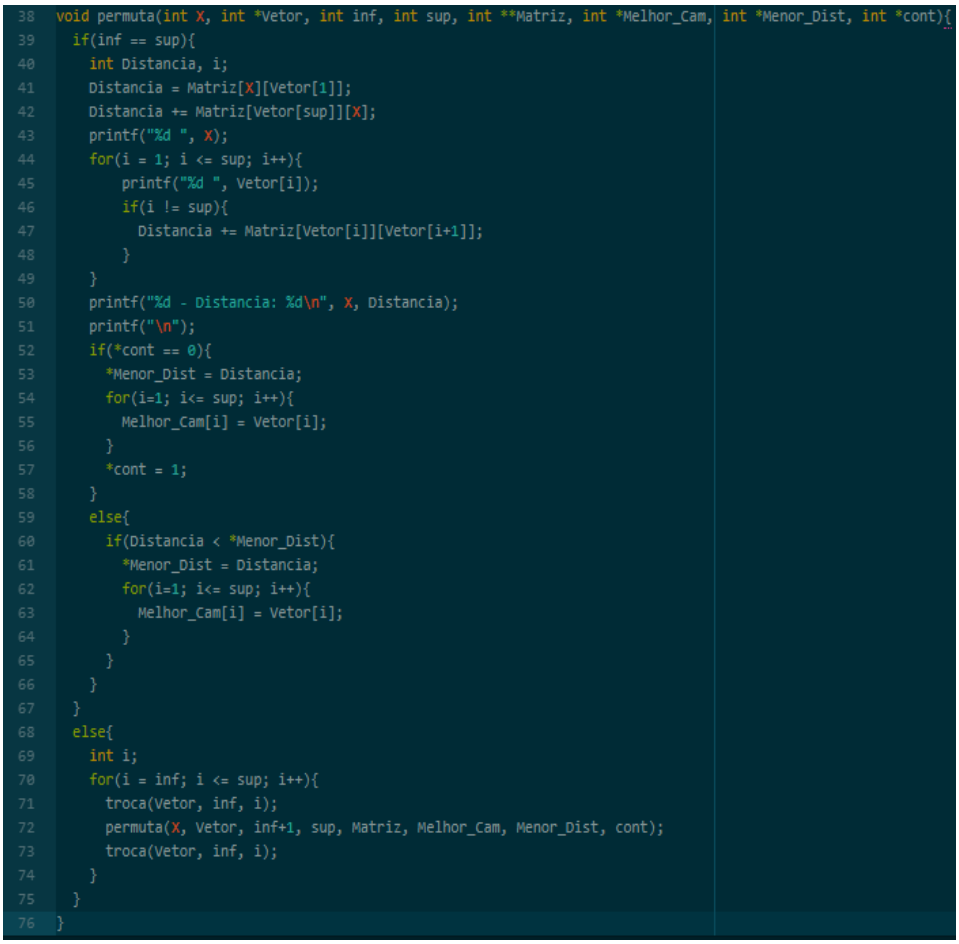
Este algoritmo funciona de forma recursiva, sendo que a cada chamada do subprograma ele realiza a troca de posição das cidades, incrementando 1 no limite inferior, de forma que a condição de parada seja quando o limite inferior for igual ao superior. Após atingir a condição de parada, ele exibe cada 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.



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.

Vale ressaltar que, durante o processo de implementação, nos deparamos com alguns impasses, sendo o mais significativo deles a forma de criação da matriz de distância e do vetor de posições. Pensamos em declarar, tanto a matriz quanto o vetor, de forma estática, visto que a utilização dos dois não era complicada, mas ao realizarmos os primeiros testes, obtivemos muitos problemas, e a forma que alcançamos para resolvê-los foi criar a matriz e o vetor de forma dinâmica, visto que com a declaração dos dois dinamicamente, não obtivemos mais problemas.

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



```c
void permuta(int X, int *Vetor, int inf, int sup, int **Matriz, int *Melhor_Cam, int *Menor_Dist, int *cont){
  if(inf == sup){
    int Distancia, i;
    Distancia = Matriz[X][Vetor[1]];
    Distancia += Matriz[Vetor[sup]][X];
    printf("%d ", X);
    for(i = 1; i <= sup; i++){
        printf("%d ", Vetor[i]);
        if(i != sup){
          Distancia += Matriz[Vetor[i]][Vetor[i+1]];
        }
    }
    printf("%d - Distancia: %d\n", X, Distancia);
    printf("\n");
    if(*cont == 0){
      *Menor_Dist = Distancia;
      for(i=1; i<= sup; i++){
        Melhor_Cam[i] = Vetor[i];
      }
      *cont = 1;
    }
    else{
      if(Distancia < *Menor_Dist){
        *Menor_Dist = Distancia;
        for(i=1; i<= sup; i++){
          Melhor_Cam[i] = Vetor[i];
        }
      }
    }
  }
  else{
    int i;
    for(i = inf; i <= sup; i++){
      troca(Vetor, inf, i);
      permuta(X, Vetor, inf+1, sup, Matriz, Melhor_Cam, Menor_Dist, cont);
      troca(Vetor, inf, i);
    }
  }
}
```

- **Dos Resultados Obtidos**

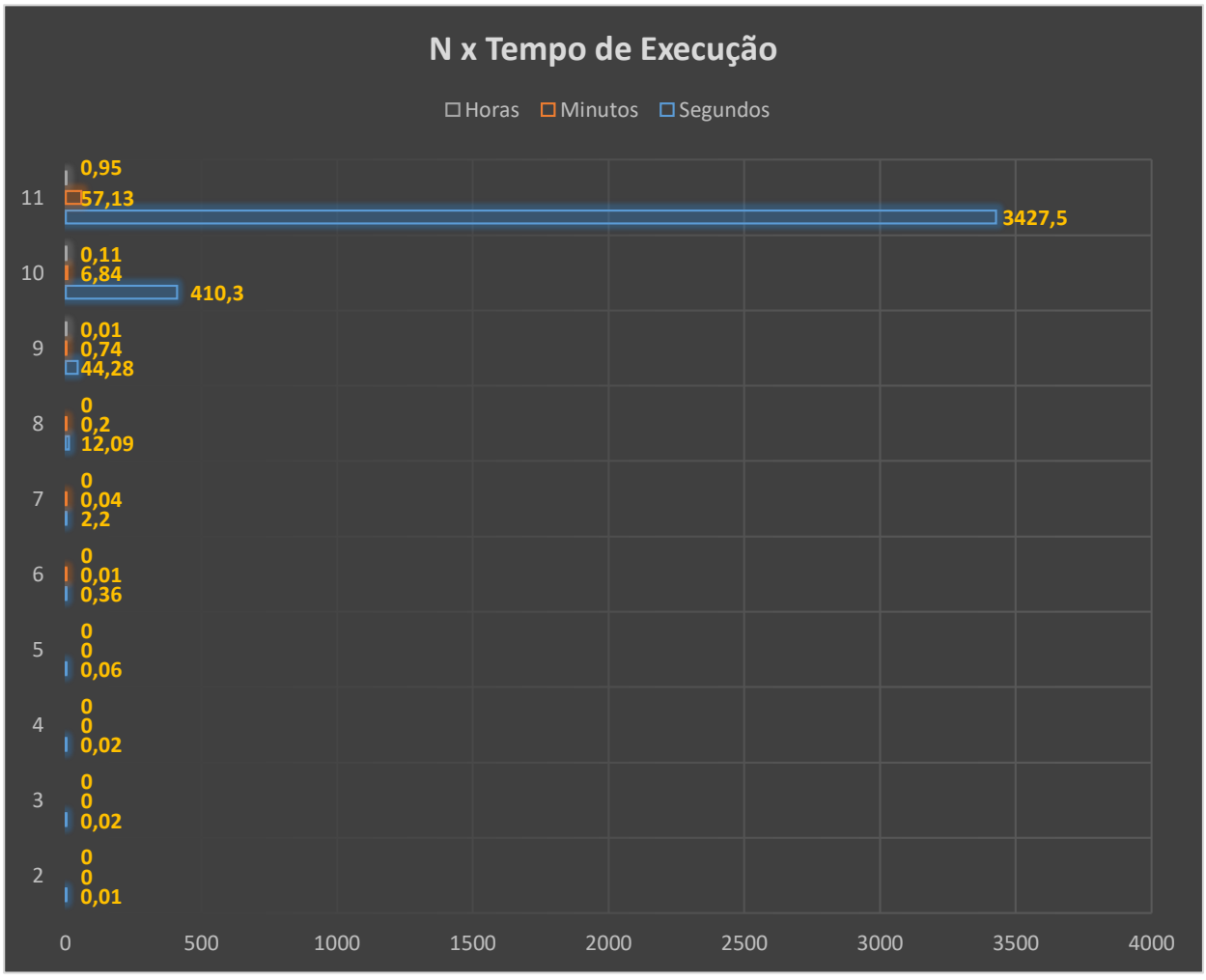
Desenvolvido o programa, partimos para a análise das entradas, começando com N=2 e incrementando 1 até chegarmos em N=11. Consideramos a matrícula de cada integrante do grupo (3465, 3488 e 3511) para todos os testes e as distâncias entre as cidades geradas aleatoriamente pela função disponível da linguagem 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: CodeBlocks::17.12;
Compilador: GNU GCC Compiler.

Tempo / N	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.