Universidade Federal de Viçosa - Campus Florestal

Trabalho prático 02 - Algoritmos e Estrutura de Dados I

Arthur De Bellis Gomes - 03503

Pablo Ferreira - 03480

Saulo Miranda Silva - 03475

Objetivo do trabalho

O objetivo do trabalho é avaliar o impacto causado pelo desempenho dos algoritmos em sua execução real. O problema adotado para essa representação foi o problema do caixeiro viajante.

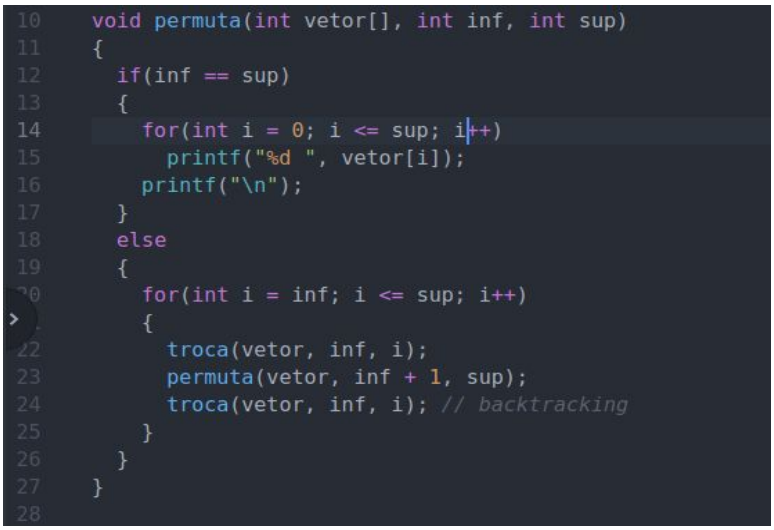
Código implementado

O código de permutação que o grupo utilizou está disponibilizado no seguinte site: https://gist.github.com/marcoscastro/60f8f82298212e267021#file-permuta-cpp.

Explicação do Código

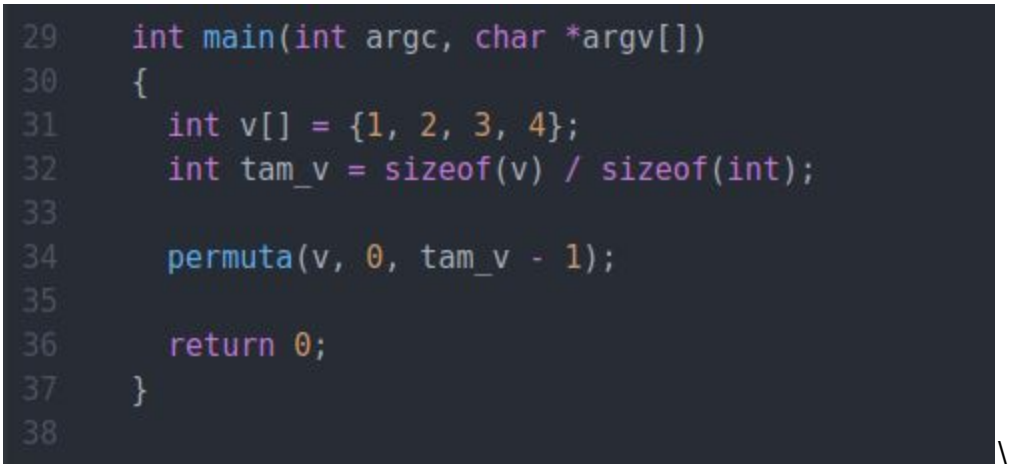
```
3   void troca(int vetor[], int i, int j)
4   {
5     int aux = vetor[i];
6     vetor[i] = vetor[j];
7     vetor[j] = aux;
8   }
9
```

Na função troca, ele troca a posição dos elementos do vetor.



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

Na função permuta, ele usa um *if* para saber se o *inf* e *sup* são iguais, caso seja, ele imprime a permutação atual. Caso não seja igual, ele usa um for e aplica a recursividade trocando e permutando os elementos do vetor.



```c
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;
}
```

\

No main, ele declara um vetor de inteiros e o preenche com alguns números, e após isso, ele declara um inteiro *“tam_v”* e usa funções para pegar o tamanho do vetor , e então ele chama a função “*permuta*” e passa como parâmetro o vetor que foi criado, 0 e o tamanho-1.

Explicação do nosso código

Como o programa calcula os custos dos caminhos:

```
void Permutacao_Permuta(int vetor[], int inf, int sup, int pPartida, int Ma
{
  int soma = 0, i = 0;
  if(inf == sup)
  {
    soma+=MatrizCidades[pPartida][vetor[0]]; //Somar a cidade inicial com a
    soma+=MatrizCidades[vetor[sup]][pPartida]; //Somar a ultima com a cidad

    printf("----------------------------------------------------------------
    printf("%d ", pPartida);

    for(i = 0; i <= sup; i++){
      if(i>0){
        soma +=MatrizCidades[vetor[i-1]][vetor[i]]; //Somar os caminhos
      }
      printf("%d ", vetor[i]);
    }
```

O inteiro **soma** é declarado dentro da função Permutacao_Permuta, que possui um if e um else, quando a condição do if é verdadeira o vetor está pronto, então uma das permutações foi feita. Com o valores do vetor é possível encontrar as distâncias armazenadas na matriz e somá-las, somando também as distâncias da cidade inicial para a primeira cidade do caminho e da última cidade do caminho para a cidade inicial.

Como escolhe o menor caminho:

```c
void Permutacao_Iniciar(int NumeroCidades, int pPartida,int MatrizCidades[NumeroCidades][NumeroCidades])
{
  int i, cont = 0, menorcaminho = 2147483647; // variavel menorcaminho com o maior numero int para poder pegar o menorc

  // Vetor com n-1 para não ter a cidade inicial
  int v[NumeroCidades-1];

  //vetor para armazenar o caminho com a menor distancia
  int caminhoMenor[NumeroCidades];

  // Gerar um vetor sem a cidade inicial
  for(i = 0; i < NumeroCidades-1; i++)
  {
    if(i == pPartida){
      cont ++;
    }
      v[i] = cont;
      cont ++;
  }

  int tam_v = sizeof(v) / sizeof(int);

  Permutacao_Permuta(v, 0, tam_v - 1, pPartida, MatrizCidades, &menorcaminho, NumeroCidades, caminhoMenor);
  printf("--------------------------------------------------------------------\n");
```

Para salvar o menor caminho dentro da função Permutacao_Iniciar declaramos o inteiro **menorcaminho** com o valor *2147483647*(maior valor positivo que um int pode armazenar) e o vetor caminhoMenor e passamos o vetor e o ponteiro para o menorcaminho como parâmetro na função Permutacao_Permuta.

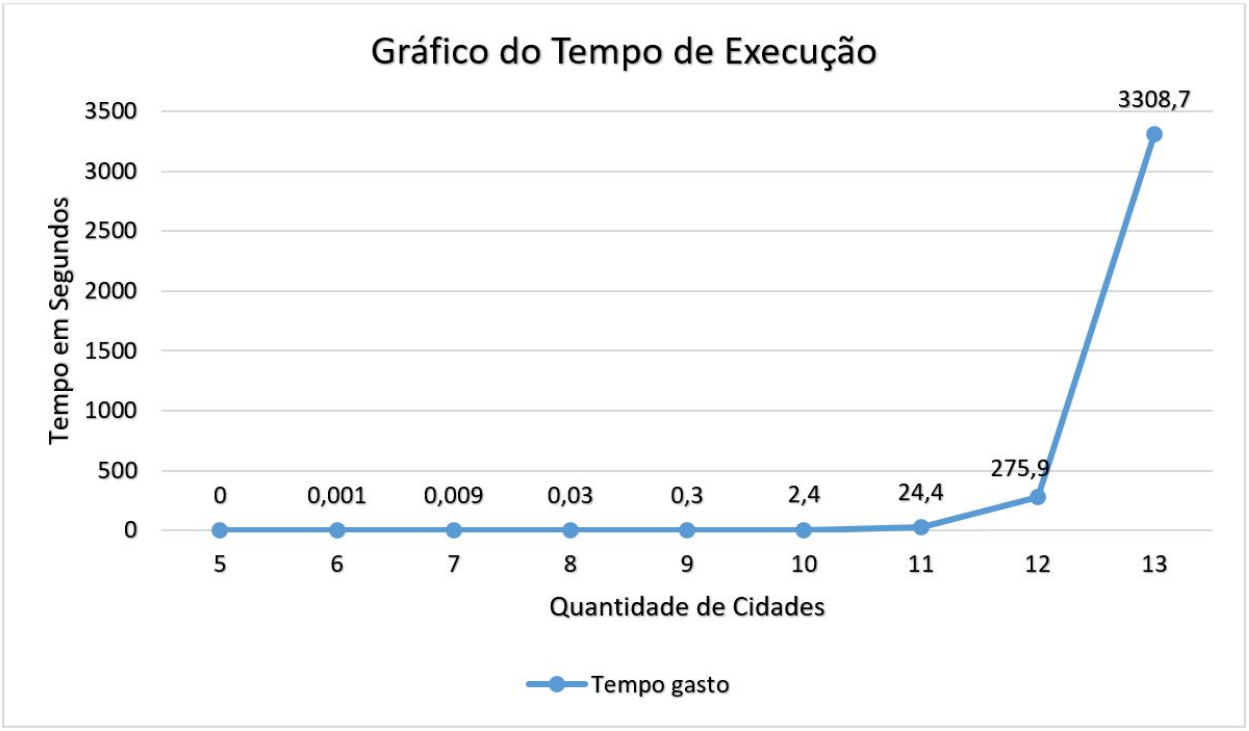
```
// compara o numero menor anterior com o
if(*menor>soma){
  *menor = soma;
  for(i = 0; i < NumeroCidades; i++)
  {
    caminhoMenor[i] = vetor[i];
  }
}
```

Nesta última função ao encontrar a distância o programa compara com a distância apontada pelo ponteiro menor(menorcaminho), se for menor do que a armazenada, menorcaminho passa a ter este valor e o vetor caminhoMenor salva o caminho com a menor distância.

Especificações da Máquina:

Tempos de execução:

Quantidade de Cidades	Tempo gasto
5	0,0004 s
6	0,001 s
7	0,009 s
8	0,03 s
9	0,3 s
10	2,4 s
11	24,4 s
12	275,9 s
13	3308,7 s



Funções

Para melhor organização do código, dividimos o programa em 6 arquivos além do *main.c,* são eles: *“Permutacao.c, Matriz.c, Menu.c”,* e seus respectivos “.h”, os arquivos têm as seguintes funções:

Permutação:

- ❏ Permutacao_Iniciar
- ❏ Permutacao_Permuta
- ❏ Permutacao_Troca
- ❏ Permutacao_SomaMatricula

Matriz:

- ❏ Matriz_Iniciar
- ❏ Matriz_Imprimir

Menu:

- ❏ Menu_Entradas
- ❏ Menu_Saida
- ❏ Menu_Confirmacao
- ❏ Menu_Arquivo

Métodos implementados

Toda edição do código dos três integrantes do grupo foi feita usando o editor de texto Atom. E assim como no último trabalho, a integração do código foi feita inteiramente pelio GitHub. A execução foi feita usando o gcc no terminal do linux.

Após certificarmos que estávamos na pasta do trabalho que contém os arquivos ".c", a pasta Sources, o seguinte comando era executado no terminal:

gcc main.c -o EXEC Permutacao.c Matriz.c Menu.c

E para executar:

./EXEC

Início da execução do programa:

Após entrar no programa, um menu mostra as opções de entrada possíveis para o usuário:

1. Modo interativo

2. Arquivo

0. Sair

Utilização do modo interativo:

Logo após a escolha desse modo pelo usuário, o programa pede a matrícula dos 3 alunos sucessivamente, após isso o usuário digita a quantidade de cidades existentes no caminho.

Utilização do modo por arquivo:

No modo por arquivo, o programa exigirá que o usuário digite o nome do arquivo que estará localizado na mesma pasta em que o programa está. Nesse arquivo estará digitado todas as funções e dados esperados para cada uma delas.

O arquivo deverá seguir esse padrão de digitação:

Exemplo de arquivo:

1812
1232
3231
3
37
89
91
56
15
15
41
84
63

Observação: O nome do arquivo precisa seguir o padrão de extensão ".txt" no final.

Exemplo: Teste.txt

Opção de saída:

Caso o consumidor do programa decida parar sua execução, ele deverá digitar "0" no menu de opções. Para se evitar paradas de execução indesejadas ou inesperadas, é exibido um campo com pedido de confirmação, este pede para usuário digitar 's' ou 'S', para parar a execução, evitando assim, certos desconfortos .

Pergunta: “Você usaria a solução desenvolvida neste trabalho?”

A solução desenvolvida é viável para transportadoras de pequeno porte, onde o número máximo de cidades percorridas pelos caminhões é 12, porém para percorrer mais cidades, o algoritmo pode ser ineficiente para achar o melhor caminho, o que por conseguinte geraria certo prejuízo para a empresa.

Conclusão

Com esse trabalho, concluímos que a forma de se fazer e implementar um algoritmo tem impactos diretos e visíveis em seu desempenho final. Além de que, algoritmos bem implementados rodam com valores semelhantes em diferentes computadores, mesmo eles tendo capacidades de memória e processamento distintos.

Dedicamos nossos agradecimentos aos monitores: Kayque Avelar e Angelo Bernar pela ajuda e apoio em questões relativas ao trabalho e a nossa professora Thais Regina (que após ler essa documentação, vai nos dar 1 ponto extra).