Universidade Federal de Viçosa

CAMPUS FLORESTAL

TRABALHO PRATICO

DESENVOLVIDO PARA DISCIPLINA CCF211
ALGORITMOS E ESTRUTURA DE DADOS 1

Pablo Miranda Batista	3482
Lucas Ranieri Oliveira Martins	3479
João Victor Magalhães Souza	3483

FLORESTAL – MG
OUTRUBRO DE 2018

Sumário

1. Objetivo e Desenvolvimento

O objetivo desse projeto foi criar um algoritmo para o problema do caixeiro viajante, porém nosso foco não em criar um algoritmo eficiente, mas sim avaliar a solução exata do problema proposto.

Inicialmente fizemos uma pesquisa para encontrar um algoritmo de calculo de permutação mais eficaz aos nossos olhos, posteriormente fizemos modificações para adequá-lo projeto. Finalmente, o restante do projeto foi criar funções para gerar a cidade de partida, calcular a distancia a ser percorrida e criar a matriz de distancias.

A maior dificuldade encontrada foi em interpretar o algoritmo de permutação que escolhemos, além de adapta-lo para nosso uso. Após resolvemos esse impasse, foi questão de tempo para concluir o projeto.

2. Algoritmos

I. Função de permutação

```
int tam_v = sizeof(v) / sizeof(int);
printf("====================================\n");
permuta(v, 0, tam_v - 1, n, matriz_rota, x, &menor_rota, vetor_rota); //4 ultimos parametros foram p/ calculo da distancia
printf("====================================\n");
```

```
void permuta(int vetor[], int inf, int sup, int n, int matriz_rota[n+1][n+1], int x, int *menor_rota, int vetor_rota[])
{
    int distancia=0;
    if(inf == sup)
    {
        printf("%d ",x);
        for(int i = 0; i <= sup; i++){
            printf("%d ", vetor[i]);
        }
        printf("%d ",x);

        verifica_menor_rota(menor_rota, vetor, n, matriz_rota, x, vetor_rota, &distancia); //funcao que compara a combinacao atual do
        // "vetor", calcula distancia e compara com o menor_rota, se for menor, ele entra no lugar dela.
        printf(" - distancia: %d\n", distancia);
    }
    else
    {
        for(int i = inf; i <= sup; i++)
        {
            troca(vetor, inf, i);
            permuta(vetor, inf + 1, sup, n, matriz_rota, x, menor_rota, vetor_rota);
            troca(vetor, inf, i); // backtracking
        }
    }
}
```

Essa função consiste em receber a matriz de distância, para geração da mesma, um vetor com ‘n’ posições que será usado para armazenar as possíveis rotas, numero de cidades, cidade inicial, e algumas variáveis para o cálculo da menor rota e armazenamento desse valor e sua rota relativa.

A função usa o vetor onde está armazenada a possível rota atual, o imprime, e depois chama a função ‘verifica_menor_rota’, onde irá calcular e imprimir a soma das distâncias da rota específica. Após isso, é conferido se a rota gerada é a menor rota, caso positivo, o algoritmo a armazena em uma variável para no final a exibirmos para o usuário. Ao voltar para função ‘permuta’, é usado à função chamada ‘troca’ que realiza, como o nome diz, a troca das posições do vetor para criar uma nova possibilidade de rota. Por fim, é chamada a função ‘permuta’ novamente, ou seja, essa e uma função com conceito de recursividade, porém ela faz todos os processos necessários no projeto.

II. Função de soma e verificação da menor rota

```
void verifica_menor_rota(int *menor_rota,int vetor[],int n,int matriz_rota[n+1][n+1],int x,int vetor_rota[], int *distancia){
    //x = numero referente a cidade de partida
    int i, j, distancia_rota=0;
    for (i = 1; i<n; i++){
        distancia_rota += matriz_rota[vetor[i-1]][vetor[i]];
    }
    distancia_rota += matriz_rota[x][vetor[0]];
    distancia_rota += matriz_rota[vetor[n-1]][x];

    if(distancia_rota < *menor_rota){
        *menor_rota = distancia_rota;
        for(i=0; i<n; i++){
            vetor_rota[i] = vetor[i];
        }
    }
    *distancia = distancia_rota;
}
```

Anteriormente citamos que o algoritmo de permutação utiliza uma função capaz de somar e decidir qual é a rota com menor distância. Essa função busca programar tal funcionalidade.

Como podemos ver, essa função acessa o vetor que tem armazenado uma rota e o utiliza para pesquisar as posições na matriz de distâncias entre as cidades e soma-las. Após isso, ele “imprime” na tela a soma e depois verifica se essa distancia é menor que a armazenada na variável para onde aponta o ponteiro ‘menor_rota’, além disso, armazena a rota em um vetor exibição da mesma.

III. Funções de manipulação do vetor

```
void preenche_vetor(int vetor[], int n, int x){                /// Gerando os numeros das cidades.
    int i, aux=0;
    for (i=0; i<n+1; i++){
        if (i != x){
            vetor[aux] = i;
            aux++;
        }
    }
}


void troca(int vetor[], int i, int j)       ///Trocando posicoes do Vetor.
{
    int aux = vetor[i];
    vetor[i] = vetor[j];
    vetor[j] = aux;
}
```

A função ‘preenche_vetor’ é utilizada para, como o nome sugere, preencher o vetor com a primeira possibilidade de rota possivel.

A função troca e utilizada para fazer a troca das posições do vetor para que ele fique com a próxima possivel rota, ocorrendo assim a permutação.

IV. Função que gera a cidade inicial

```
int matricula (char *m1, char *m2, char *m3, int n){
    int x1, x2, x3, final;
    x1 = (((m1[0] - '0')*1) + ((m1[1] - '0')*1) + ((m1[2] - '0')*1) + ((m1[3] - '0')*1));
    x2 = (((m2[0] - '0')*1) + ((m2[1] - '0')*1) + ((m2[2] - '0')*1) + ((m2[3] - '0')*1));
    x3 = (((m3[0] - '0')*1) + ((m3[1] - '0')*1) + ((m3[2] - '0')*1) + ((m3[3] - '0')*1));
    final = (x1 + x2 + x3) % n;
    return final;
}
```

Essa função consiste em receber as três matriculas digitadas pelo usuário que estão em formato string. Sua funcionalidade é fazer a conversão de cada caractere das strings para valor numérico e realizar a soma de todos eles. Posteriormente retorna o resto da divisão dessa soma pelo numero de cidades.

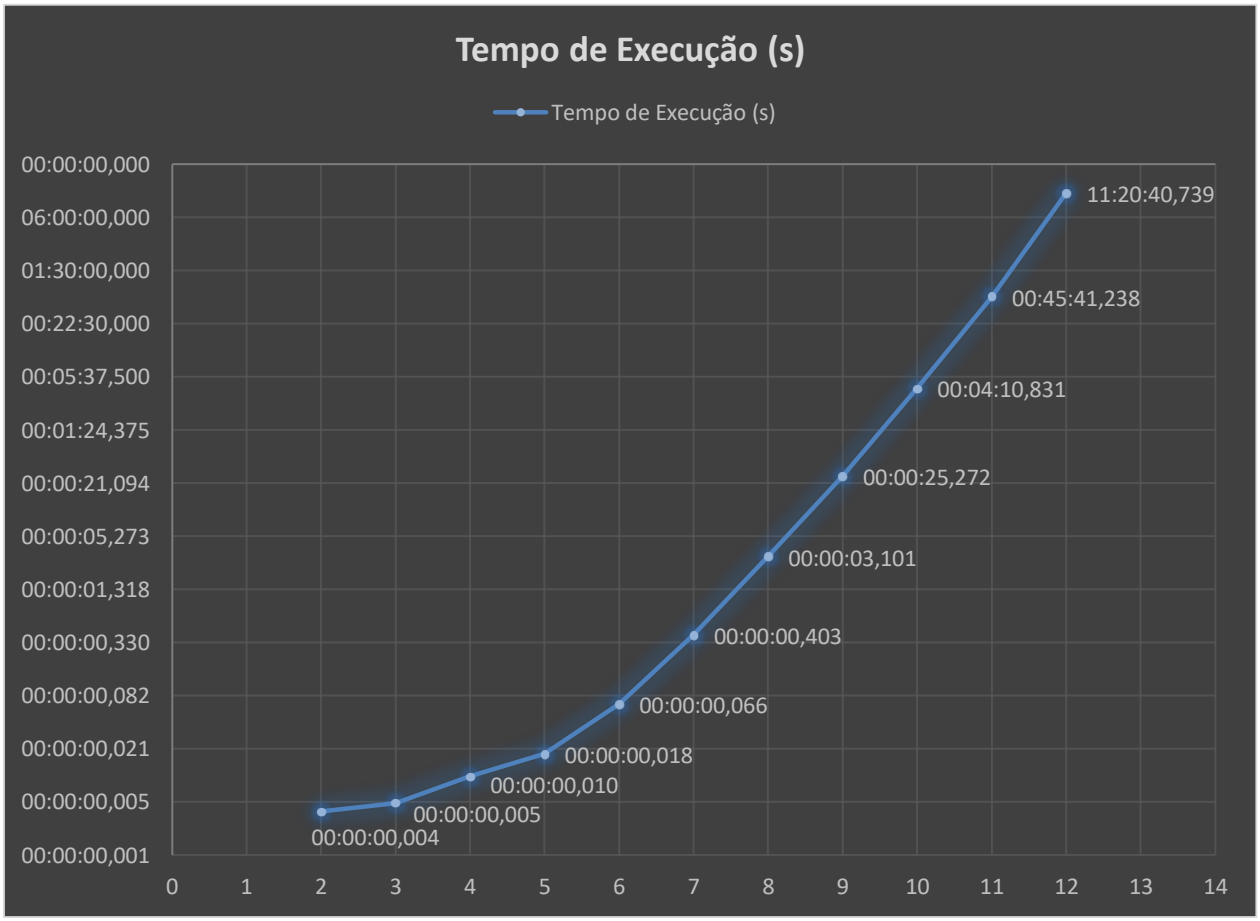
3. Análise do Tempo de Execução

Para fazer a análise de tempo gasto de execução foi utilizado um notebook com as seguintes configurações:

- Processador Intel Core i3, 2.27 GHz;
- 8,00 GB de memória RAM;
- Disco Rígido de 1 TB.

Mostraremos uma tabela e gráfico que demonstra quanto tempo o algoritmo leva para ser executado para entrada de ‘n’ cidades:

Número de cidades (n)	Tempo de Execução (s)
2	0.004 s
3	0.005 s
4	0.010 s
5	0.018 s
6	0.066 s
7	0.403 s
8	3.101 s
9	25.272 s
10	250.831 s
11	2741.238 s
12	40840.739 s



Como podemos ver no gráfico, o algoritmo tem um ótimo tempo de execução até o tamanho da entrada igual a nove, após isso o algoritmo começa a ter um tempo elevado, onde esse tempo de execução fica extremamente alto a partir de uma entrada de tamanho onze. Só conseguimos executar o teste de tempo de execução até tamanho doze, acima disso o tempo é extremamente alto, e não teríamos tempo hábil para fazer tal teste. Mas fizemos uma pequena estimativa de quanto seria o tempo, ela nos diz que a partir de treze ou mais cidades de entrada o algoritmo irá ter um tempo de execução 10x maior do que o tempo da entrada anterior. Por exemplo, se a entrada for treze, o algoritmo terá um tempo de execução de mais o menos quatro dias e meio, e se a entrada for quatorze, ele irá nos retornar um resultado em mais o menos quarenta dias.

4. Conclusão

Nesse projeto obtivemos a confirmação do que foi visto sala de aula na disciplina de AEDS, que nem tudo que está funcionando é eficiente e utilizável. Percebemos a real importância de se planejar e desenvolver algoritmos para cada caso, pois muitas vezes criamos algoritmos usando nossa subjetividade, mas quando vamos testa-lo no ambiente do cliente, o resultado não satisfatório. Sempre devemos fazer uma pesquisa de campo no local onde será implementado uma solução, para conhecermos onde e por quem o algoritmo será utilizado.

Neste projeto foi proposto que respondêssemos uma pergunta ao fim do desenvolvimento do algoritmo **(Caso você fosse contratado por uma transportadora para desenvolver um sistema que encontrasse a menor rota a ser percorrida por seus caminhões, saindo e chegando do galpão de estoque e percorrendo uma única vez um conjunto de N localidades, você utilizaria a solução desenvolvida neste trabalho? Justifique a resposta)**. A resposta para essa pergunta é claramente não, pois nosso algoritmo e razoável para casos com numero ‘n’ de entradas baixo. Ao aumentar o número de entrada seu tempo de execução fica muito longo, e uma empresa de transportes busca velocidade e economia. Além disso, existem muitos outros dados que devem ser avaliados para se definir a melhor rota, como por exemplo, transito interno da cidade, urgência de entrega, horário de entrega para cada cliente etc. Visto isso, nosso algoritmo seria catastrófico em qualquer empresa de transportes, porém ele serve como uma ótima base para se iniciar esse projeto, fazendo uma análise das consequências de um algoritmo “exato”, mas não eficiente.

Finalmente, o trabalho prático nos ajudou a mostrar mais uma aplicação e a importância da matéria que estamos estudando em sala de aula. Esperamos que os próximos projetos, e matérias, sigam essa linha, pois exercitar o que vimos em sala é importante para aprender a desenvolver algoritmos conscientes.

5. Referências

Neste link está o algoritmo que utilizamos para fazer nosso projeto:

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