Universidade Federal do Sul e Sudeste do Pará
Faculdade de Computação e Engenharia Elétrica
Curso de Engenharia da Computação

Juliana Batista da Silva
Iago Costa das Flores
Warley Rabelo Galvão

Complexidade de Algoritmos
Trabalho Final - Prova 3

Marabá
2021

Juliana Batista da Silva
Iago Costa das Flores
Warley Rabelo Galvão

Complexidade de Algoritmos
Trabalho Final - Prova 3

Relatório apresentado no curso de Engenharia da Computação, turma de 2018 como obtenção de nota parcial na disciplina de Complexidade de algoritmo, ministrada pelo Professor Dr. Manoel Ribeiro.

Marabá
2021

UNIFESSPA

Sumário

1 - Introdução

Neste trabalho iremos apresentar uma solução que usa a técnica de algoritmos gulosos para resolver o problema. Os algoritmos gulosos como o nome indica tem por padrão escolher em cada iteração, a resultado mais “apetitoso” para compor a solução do algoritmo. Além disso o algoritmo é conhecido por ser de certa forma “míope” ele toma decisões com base na iteração atual e não calcula as consequências futuras dessa escolha e também não se arrepende das escolhas já feitas.

Será apresentado uma solução gulosa para o algoritmo de número mínimo de abastecimentos descrito a seguir: Quero dirigir um carro de uma cidade A a uma cidade B ao longo de uma rodovia. O tanque de combustível tem capacidade suficiente para cobrir N quilômetros. Mapa da rodovia indica a localização dos postos de combustível. Dê um algoritmo que garanta uma viagem com número mínimo de abastecimentos.

2 - Desenvolvimento

Primeiramente, na resolução do problema é preciso compreender as principais variáveis que definem o problema do mínimo de abastecimento durante determinado percurso. As variáveis têm relação direta com o carro usado e seu combustível, são elas consumo de combustível por quilômetro, a capacidade em litros do tanque de combustível, o percurso a percorrer, as posições em quilômetros dos postos de combustível e a quantidade de postos de combustível dentro do percurso.

Após determinadas variáveis podemos passar para lógica de execução do algoritmo. Ele pode ser resolvido com um for central que percorre o percurso de um em um quilômetro. Dentro dele e feito um outro for para verificar se o motorista está ou não no mesmo quilômetro que o posto de gasolina se estiver ele verifica se seu combustível consegue entrar no próximo posto de gasolina através do mapa se conseguir ele não abastece e segue adiante, caso não consiga ele abastece naquele posto atual e segue a viagem dessa forma ele vai tentar entrar no posto de combustível apenas quando realmente for necessário caso contrário ele segue viagem.

Como é possível notar na explicação da resolução, temos o algoritmo sempre fazendo escolhas a cada interação, buscando a melhor opção do momento sem olhar para as consequências futuras e sem se arrepender das escolhas feitas.

2.1 Apresentação do Algoritmo

Na figura 01 temos o código fonte do algoritmo com suas linhas comentadas e explicadas.

```c
#include <stdlib.h>
#include <stdio.h>

// Quero dirigir um carro de uma cidade A a uma cidade B ao longo de uma rodovia.
// O tanque de combustível tem capacidade suficiente para cobrir N quilômetros.
// Mapa da rodovia indica a localização dos postos de combustível.
// Dê um algoritmo que garanta uma viagem com número mínimo de re-abastecimentos.

int minimo_abastecimento // função que calcula o número mínimo de abastecimentos
    (
        int consumo_litro_por_km,              // Consumo de combustível por quilômetro
        int tanque_km,                         // Capacidade do tanque
        int percurso_km,                       // Percurso a percorrer
        int *postos_combustiveis_km,           // Posições dos postos de combustível
        int postos_combustiveis_km_quantidade // Quantidade de postos de combustível
    );

int consumo_entre_postos // função que calcula o consumo entre postos de combustível
    (
        int posto_proximo,        // Posição do posto de combustível mais próximo
        int posto_atual,          // Posição do posto de combustível atual
        int consumo_litro_por_km // Consumo de combustível por quilômetro
    );                            // Consumo entre postos

int main(int argc, char const *argv[])
{
```

```
   int vetor[] = {2, 6, 8, 12, 15, 19, 22, 25, 28, 35, 39, 41, 43, 47,
49};
   int consumo_litro_por_km = 1;
   int tanque_km = 9;
   int percurso_km = 50;
   int tamanho_vetor = (sizeof(vetor) / sizeof(vetor[0]));

   printf("Inicio do percurso \n");
   // int consumo_litro_por_km,  // Consumo de combustível por
quilômetro
   // int tanque_km, // Capacidade do tanque
   // int percurso_km, // Percurso a percorrer
   // int *postos_combustiveis_km, // Posições dos postos de
combustível
   // int postos_combustiveis_km_quantidade // Quantidade de postos de
combustível
   int abastecimentos = minimo_abastecimentos(consumo_litro_por_km,
tanque_km, percurso_km, vetor, tamanho_vetor);
   printf("\nFim do percurso!!! \n \n");
   printf("Consumo litro/km foi %d, Capacidade do tanque era %d
litros\ne o tamanho do percurso era %d Kms", consumo_litro_por_km,
tanque_km, percurso_km);
   printf("\nO número de abastecimentos foi %d, de %d postos de
combustível. \n \n", abastecimentos, tamanho_vetor);

   return 0;
}

int minimo_abastecimentos // função que calcula o número mínimo de
abastecimentos
   (
       int consumo_litro_por_km,             // Consumo de combustível
por quilômetro
       int tanque_km,                        // Capacidade do tanque
       int percurso_km,                      // Percurso a percorrer
       int *postos_combustiveis_km,          // Posições dos postos de
combustível
       int postos_combustiveis_km_quantidade // Quantidade de postos de
combustível
   )
{
   int tanque_restante_km = 0,         // Capacidade do tanque restante
```

```
       posto_combustivel_atual_km = 0, // Posição atual do posto de
combustível
       contador_abastecimentos = 0,    // Contador de abastecimentos
       percurso_restante_km = 0,       // Percurso a percorrer restante
       consumo_entre_postos_atual = 0, // Consumo entre postos atual
       i = 0;                          // Número de postos de
combustível
   tanque_restante_km = tanque_km;     // inicializa o tanque restante
com o tanque inicial

   // imprimir numero de postos de combustiveis
   printf("Números de postos %d \n",
postos_combustiveis_km_quantidade);
   for (int posicao_atual_km = 0; posicao_atual_km < percurso_km;
posicao_atual_km++) // percorre o percurso
   {
       printf("Tanque restante: %d \n", tanque_restante_km);
// Mostra o tanque restante
       printf("Posto de combustível atual: %d \n",
posto_combustivel_atual_km + 1); // Mostra o posto de combustível atual
       printf("Posição atual do motorista: %d \n", posicao_atual_km);
// Mostra a posição atual do posto de combustível
       percurso_restante_km = percurso_km - posicao_atual_km;
// Calcula o percurso restante
       printf("Percurso restante: %d \n", percurso_restante_km);
// Mostra o percurso restante
       printf("Percurso total: %d \n", percurso_km);
// Mostra o percurso total
       printf("======  %d (km rodados) ====== \n \n",
posicao_atual_km);            //Iterador

       for (i = 0; i < postos_combustiveis_km_quantidade; i++) //
percorre todos os postos de combustiveis
       {
           if (posicao_atual_km == postos_combustiveis_km[i]) // Se
encontrou o posto de combustível
           {
               if (i != (postos_combustiveis_km_quantidade - 1)) // Se
não for o último posto de combustível
               {
                   consumo_entre_postos_atual =
consumo_entre_postos(postos_combustiveis_km[i + 1],
postos_combustiveis_km[i], consumo_litro_por_km);
```

```
                    if (tanque_restante_km > consumo_entre_postos_atual)
// Se o tanque do motorista está suficiente para o consumo
                    {
                        printf("Motorista não abasteceu no posto %d \n",
i); // Não abasteceu
                        posto_combustivel_atual_km = i;
// reinicia o posto de combustível
                    }
                    if (tanque_restante_km < consumo_entre_postos_atual)
// Se o tanque do motorista não está suficiente para o consumo
                    {
                        tanque_restante_km = tanque_km;
// Reinicia o tanque
                        posto_combustivel_atual_km = i;
// reinicia o posto de combustível
                        printf("Motorista abasteceu no posto %d \n", i);
// Abastece no posto de combustível
                        contador_abastecimentos += 1;
// Incrementa o contador de abastecimentos
                    }
                }
                if (i == (postos_combustiveis_km_quantidade - 1)) // Se
for o último posto de combustível
                {
                    consumo_entre_postos_atual =
consumo_entre_postos(percurso_km, postos_combustiveis_km[i],
consumo_litro_por_km);

                    if (tanque_restante_km > consumo_entre_postos_atual)
// Se o tanque do motorista está suficiente para o consumo
                    {
                        printf("Motorista não abasteceu no posto %d \n",
i); // Não abasteceu
                        posto_combustivel_atual_km = i;
// reinicia o posto de combustível
                    }
                    if (tanque_restante_km < consumo_entre_postos_atual)
// Se o tanque do motorista não está suficiente para o consumo
                    {
                        tanque_restante_km = tanque_km;
// Reinicia o tanque
                        posto_combustivel_atual_km = i;
// seta posto de combustível
```

```
                        printf("Motorista abasteceu no posto %d \n", i);
// Abastece no posto de combustível
                        contador_abastecimentos += 1;
// Incrementa o contador de abastecimentos
                    }
                }
            }
        }

        if (tanque_restante_km == 0) // Se o tanque estiver vazio
        {
            printf("Gasolina acabou!! \n"); // A gasolina acabou
            return contador_abastecimentos; // Retorna o contador de
abastecimentos
        }

        tanque_restante_km = tanque_restante_km - (consumo_litro_por_km
* 1); // Calcula o tanque restante
    }
    return contador_abastecimentos; // Retorna o contador de
abastecimentos
}

int consumo_entre_postos // Função que calcula o consumo entre postos
de combustível
    (
        int posto_proximo,        // Posto de combustível próximo
        int posto_atual,          // Posto de combustível atual
        int consumo_litro_por_km // Consumo de litros por km
    )
{
    return (posto_proximo - posto_atual) * consumo_litro_por_km; //
Calcula o consumo entre postos
}
```

Figura 01: Código fonte da solução.

Na figura 02 temos o exemplo de resultado da execução do script com os respectivos dados utilizados.

```
======  45 (km rodados) ======

Tanque restante: 6
Posto de combustível atual: 13
Posição atual do motorista: 46
Percurso restante: 4
Percurso total: 50
======  46 (km rodados) ======

Tanque restante: 5
Posto de combustível atual: 13
Posição atual do motorista: 47
Percurso restante: 3
Percurso total: 50
======  47 (km rodados) ======

Motorista não abasteceu no posto 13
Tanque restante: 4
Posto de combustível atual: 14
Posição atual do motorista: 48
Percurso restante: 2
Percurso total: 50
======  48 (km rodados) ======

Tanque restante: 3
Posto de combustível atual: 14
Posição atual do motorista: 49
Percurso restante: 1
Percurso total: 50
======  49 (km rodados) ======

Motorista não abasteceu no posto 14

Fim do percurso!!!

Consumo litro/km foi 1, Capacidade do tanque era 9 litros
e o tamanho do percurso era 50 Kms
O número de abastecimentos foi 6, de 15 postos de combustível.
```

Figura 0: Exemplo de resultado do script.

3 - Conclusão

Após a implementação do código foi realizada uma bateria de testes e neles verificam-se que o código fonte está calculando corretamente as variáveis de número mínimo de abastecimentos, consumo entre postos de combustível, a capacidade do tanque de combustível após os abastecimentos, percurso total, percurso restante e consumo de litro/km ao final do percurso. Dessa forma, verifica-se que estão sendo alcançados os resultados esperados, pois essas variáveis são de suma importância para que não haja erros nos cálculos matemáticos evitando que o motorista fique no prego no meio de uma rodovia ou via pública e garantindo que o percurso seja completo com o número mínimo de abastecimentos necessário.

Além disso, esse protótipo desenvolvido foi de grande valia para colocarmos em prática o conhecimento que foi passado em sala de aula pelo Docente da disciplina possibilitando um entendimento mais abrangente de algoritmos gulosos e garantindo o aprendizado dos discentes, visto que o trabalho prático força os discentes a absorver o conhecimento de forma significativa para poder ter mais facilidade no desenvolvimento do código fonte.


UNIFESSPA