PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL – PUCRS

ENGENHARIA DE SOFTWARE / BACHARELADO

CAROLINA FERREIRA | MATEUS CAÇABUENA | LUIZA HELLER

TRABALHO 1 – EXERCÍCIO GREEDY

Rally no Deserto de Dakkar

Porto Alegre – RS

2024/2

Sumário

1. O Problema 3
   1.1 Premissas do Problema 3
   1.2 Objetivo 3
2. O Algoritmo 4
   2.1 Pseudocódigo 5
3. Análise do Algoritmo 6
   3.1 Correção do Algoritmo 6
   3.2 Optimalidade do algoritmo 6
   3.3 Complexidade de Tempo do Algoritmo 6
   3.4 Conclusão da Análise do Algoritmo 7
4. Implementação e Tempo de Execução 8

1. O Problema

O objetivo deste trabalho é aplicar uma estratégia de algoritmo greedy para otimizar o percurso em um rally no deserto de Dakkar, de forma a minimizar o número de paradas para descanso. As condições da corrida estipulam que o time de competidores só pode realizar a viagem durante o dia, e deve fazer paradas estratégicas em pontos ao longo da trilha para descansar antes de continuar o trajeto.

1.1 Premissas do Problema

Para modelar a trilha, iremos assumir os seguintes parâmetros:

- **Comprimento total da trilha (L);**
- **Distância máxima diária (d);**
- **Pontos de Parada.**

1.2 Objetivo

Nosso objetivo com esse trabalho é determinar a sequência ótima de paradas que irá minimizar o número de paradas para descanso necessárias para que a equipe complete o percurso escolhido pelo usuário, respeitando as restrições impostas.

2. O Algoritmo

O algoritmo que foi desenvolvido segue uma abordagem greedy, para decidir se, a cada ponto de parada, a equipe consegue avaçar para o próximo ponto antes do anoitecer. A forma como as decisões são feitas é a seguinte:

1. **Definindo o próximo ponto de parada:** No início de cada iteração, vamos definir a variável *proximaParada*, que irá representar a próxima posição de parada no array *pontosParada*, e a variável *distanciaProximaParada*, que vai calcular a distância entre o ponto atual (*posicaoAtual*) e *proximaParada*.
2. **Continuar ao próximo ponto de parada:** Se a distância entre o ponto atual e o próximo ponto de parada for menor ou igual à distância máxima diária *d*, a equipe continua dirigindo.

   Em termos mais técnicos:
   - Teremos uma condição *if (distanciaProximaParada <= d)* que verifica se a equipe consegue alcançar o próximo ponto antes de anoitecer.
   - Caso seja possível, a equipe vai "continuar dirigindo", logo, a variável *posicaoAtual* vai ser atualizada para *proximaParada*.
   - O código irá imprimir uma mensagem: *"Continuamos até o ponto de parada em " + posicaoAtual + " km. "*, indicando que a equipe avaçou para o próximo ponto.
3. **Acampar no ponto atual:** Se a distância até o próximo ponto for maior que *d,* a equipe para e acampa no ponto atual. No dia seguinte, o algoritmo irá recomeçar a contagem de distância a partir desse ponto.

   Em termos mais técnicos:
   - Se a condição anterior não for atendida, entendemos que a distância ao próximo ponto é maior que *d.* Nesse caso, a equipe tem que acampar no ponto atual.
   - A mensagem: *"*Acampamos no ponto de parada em " + posicaoAtual + " km, a distância até o próximo ponto é de " + distanciaProximaParada + " km*"*
   - será impressa, e a variável *numParadas* será incrementado, de forma a registrar a parada realizada.

- Para garantirmos que no dia seguinte a equipe avalie o ponto atual novamente, utilizamos *i--*, para fazer o loop reavaliar o ponto atual na próxima iteração.

Esse processo vai se repetir até a equipe chegar ao ponto final da trilha, ou até não ser mais possível avançar.

2.1 Pseudocódigo

1. Inicialize *posicaoAtual = 0* e *numParadas = 0*.
2. Para cada ponto de parada *ponto[i]* em *pontosParada*:
    a. Calcule a distância *distanciaProximaParada* entre *posicaoAtual* e *ponto[i]*.
    b. Se *distanciaProximaParada <= d*, avance até *ponto[i]*.
    c. Caso contrário, acampe no ponto atual e aumente *numParadas*.
3. Após o loop, verifique se é possível alcançar o ponto final *L* a partir da última posição.
4. Se sim, finalize o rally. Caso contrário, indique que não foi possível completar o percurso.

3. Análise do Algoritmo

3.1 Correção do Algoritmo

Para garantirmos que o algoritmo é correto, precisamos mostrar que ele sempre vai produzir uma solução que vai cobrir a trilha de L quilometros sem ultrapassar o limite de distância diária d em nenhum ponto. Essa correção vai se basear nas premissas que foram fornecidas e na decisão local de avançar até o ponto mais distante possível no dia. Pelo enunciado, sabemos que é sempre possível avançar até algum ponto dentro do limite d, e que o algoritmo nunca vai ficar entre dois pontos distantes demais para serem alcançados no mesmo dia. Outro ponto muito importante para a correção é a lógica de parar ou avançar. Em cada iteração, o algoritmo vai verificar se a distância até o próximo ponto é menor ou igual a d. Se sim, ele vai avançar para esse ponto, e caso contrário, ele acampa. Por causa dessa abordagem greedy, asseguramos que cada dia de viagem será maximizado em termos de distância percorrida, e esse comportamento vai ser seguido até o fim do percurso, ou até o algoritmo detectar que é impossível alcançar o próximo ponto dentro do limite d.

3.2 Optimalidade do algoritmo

Para mostrarmos que o algoritmo é ótimo, temos que provar que ele minimiza o número de paradas necessárias para concluir a trilha L. O algoritmo está sendo baseado na estratégia greedy, ou seja, vai sempre ir até o ponto mais distante possóvel, dentro do limite diário de d km, minimizando, assim, o número de paradas. Essa estratégia é ótima para esse problema, visto que se o algoritmo optasse por parar em pontos intermediários sem necessidade, ao invés de ir até o mais distante possível, ele iria adicionar paradas desnecessárias, contradizendo o objetivo de minimizar paradas.

3.3 Complexidade de Tempo do Algoritmo

A complexidade vai depender principalmente do número de pontos de parada *n,* onde cada ponto representa uma posição ao longo da trilha L. Em termos mais técnicos, o algoritmo é executado em tempo **O(n)**, uma vez que ele percorre a lista de pontos de parada somente uma vez, realizando uma comparação para decidir entre avançar ou parar a cada ponto.

1. **Operação de Verificação do Próximo Ponto**: Em cada iteração, o algoritmo irá calcular a distância para o próximo ponto e vai decidir se ela permite avançar ou se vai ser necessário acampar. Em termos de complexidade, cada uma dessas operações é constante, ou seja, **O(1)**.
2. **Iterações Lineares**: Pela lista ser percorrida apenas uma vez, e não tendo loops aninhados, o algoritmo tem uma complexidade linear em relação ao número de pontos de parada, o que é de grande eficiência para este problema, já que, mesmo com um grande número de pontos, o tempo de execução vai permanecer rápido.

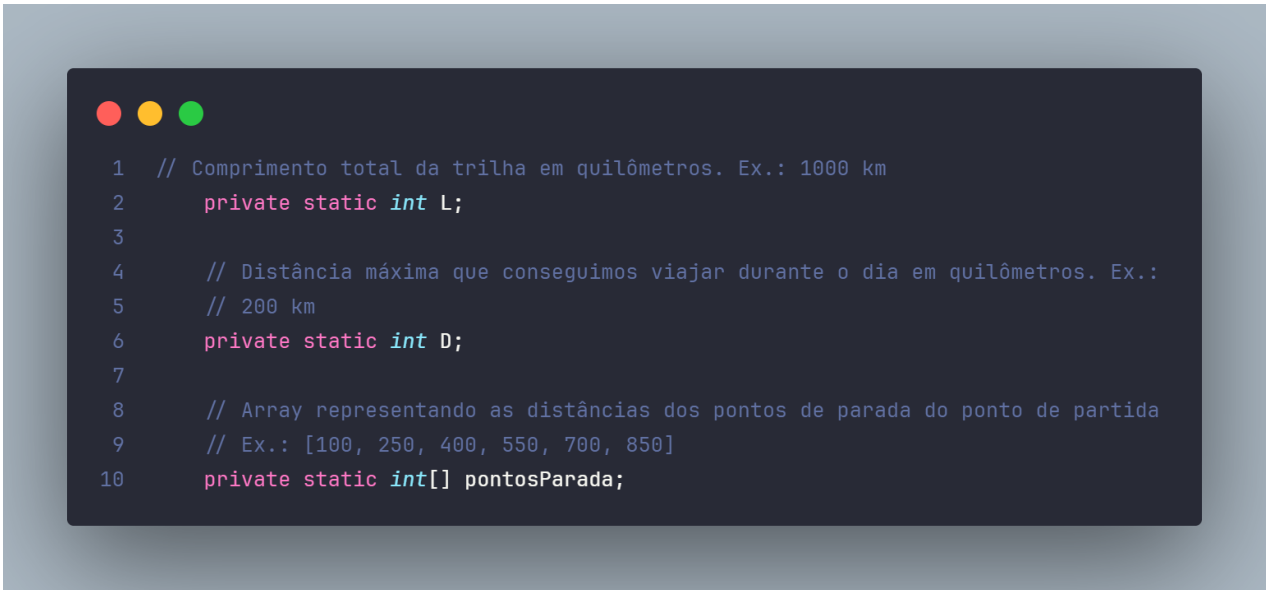
3.4 Conclusão da Análise do Algoritmo

O algoritmo greedy proposto é eficiente, correto e ótimo para resolver esse problema de minimizar o número de paradas no rally. Sua correção e optimalidade partem da lógica de avançar o máximo que for possível em cada dia, respeitando o limite diário e assegurando o menor número de paradas. A complexidade **O(n)** nos mostra que o algoritmo é adequado para percursos com grande quantidade de pontos de parada. Todos esses fatores agregam para a entrega de um algoritmo de alta eficiência para solução do problema envolvido no Trabalho 1.

4. Implementação e Tempo de Execução

Nossa implementação do algoritmo foi realizada utilizando a linguagem Java e considerando as condições apresentadas no enunciado do problema. Abaixo segue prints para o código final do trabalho, que pode ser encontrado no GitHub, através do link.

https://github.com/mateuscacabuenaPUCRS/ProjetoOtimizacaoDeAlgoritmos/tree/main/Trabalho1



```java
// Comprimento total da trilha em quilômetros. Ex.: 1000 km
    private static int L;

    // Distância máxima que conseguimos viajar durante o dia em quilômetros. Ex.:
    // 200 km
    private static int D;

    // Array representando as distâncias dos pontos de parada do ponto de partida
    // Ex.: [100, 250, 400, 550, 700, 850]
    private static int[] pontosParada;
```

```java
public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            try {
                System.out.print("Digite o comprimento total da trilha (L): ");
                L = scanner.nextInt();

                System.out.print("Digite a distância máxima que conseguimos viajar durante o dia (D): ");
                D = scanner.nextInt();

                System.out.print("Digite o número de pontos de parada: ");
                int numPontos = scanner.nextInt();
                pontosParada = new int[numPontos];

                System.out.println("Digite as distâncias dos pontos de parada do ponto de partida:");
                for (int i = 0; i < numPontos; i++) {
                    pontosParada[i] = scanner.nextInt();
                }

                long inicio = System.nanoTime();

                verificarParadas();

                long fim = System.nanoTime();

                long duracao = (fim - inicio) / 1_000_000;
                System.out.println("\nTempo de execuçao: " + duracao + " ms");

                break; // Saia do loop se tudo estiver correto

            } catch (InputMismatchException e) {
                System.out.println("Por favor, insira apenas valores numéricos.");
                scanner.next(); // Limpa a entrada inválida
            }
        }
    }

```

```java
public static void verificarParadas() {
        int posicaoAtual = 0;
        int numParadas = 0;
        int distanciaDeHoje = 0;

        System.out.println("\n\nIniciando rally:");

        for (int i = 0; i < pontosParada.length; i++) {
            int proximaParada = pontosParada[i];
            int distanciaProximaParada = proximaParada - posicaoAtual;
            int distanciaDisponivel = D - distanciaDeHoje;

            if (distanciaProximaParada > D) {
                System.out.println(
                        "\nPróxima parada inválida: distância da próxima parada é maior que a distância que conseguimos viajar por dia.");
                break;
            }

            System.out.println("\nPosiçao atual: " + posicaoAtual + " km");
            System.out.println("Próxima parada: " + proximaParada + " km");
            System.out.println("Distância disponível no dia: " + distanciaDisponivel + " km.");

            // Verifica se a próxima parada está dentro da distância máxima diária
            if (distanciaDisponivel ⩾ distanciaProximaParada) {
                posicaoAtual = proximaParada;
                distanciaDeHoje += distanciaProximaParada;
                System.out.println("*Continuamos até o ponto de parada em " + posicaoAtual + " km.*");
            } else {
                System.out.println("*Acampamos no ponto de parada em " + posicaoAtual
                        + " km, a distância até o próximo ponto é de " + distanciaProximaParada + " km*");
                distanciaDeHoje = 0;
                numParadas++;
                i--;
            }
        }

        // Verifica se chegamos ao final do rally
        if (posicaoAtual == L) {
            System.out.println("Rally completado com sucesso com " + numParadas + " paradas para acampar.");
        } else if (L - posicaoAtual ⩽ D) {
            System.out.println("-------------------------------------------------------------");
            System.out.println("Rally completado com sucesso no ponto final em " + L + " km.");
            System.out.println("Foram feitas " + numParadas + " paradas.");
            System.out.println("-------------------------------------------------------------");

        } else {
            System.out.println("Não foi possível completar o rally.");
        }
    }
```