Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) – Escola Politécnica
Disciplina de Projeto e Otimização de Algoritmos
Professor: Rafael S. Silva
Aluno: Ricardo B. Süffert

# Exercício "*Greedy*" (opcional)

## 1. O Problema:

O problema proposto para o exercício consiste na análise e na implementação de um algoritmo *greedy* que deve auxiliar, no contexto de um *rally*, a determinar quantos pontos de parada (acampamentos) a equipe pode avançar durante o dia, sendo que o regulamento da competição estabelece que pode-se viajar somente durante o dia e, à noite, os competidores devem acampar em algum ponto de parada. Naturalmente, deseja-se avançar o máximo possível durante o dia, a fim de chegar o quanto antes ao final da competição. O algoritmo deverá auxiliar a mostrar até qual ponto de parada os competidores da equipe devem avançar antes de pegar a noite na estrada.

---

## 2. O Algoritmo:

O algoritmo pensado é expresso da seguinte maneira no enunciado do exercício:

> *"Cada vez que vocês chegarem a um potencial ponto de parada, vocês determinam se conseguem chegar no próximo ponto de parada antes do anoitecer. Se vocês conseguirem, vocês continuam dirigindo; caso não consigam, vocês param e acampam no ponto atual."*

Em inglês estruturado, obtém-se algo similar a isso:

```
while the rally isn't over
      when a camping is reached
            if it is possible to reach the next camping before night
                  keep driving until the next camping is reached
            else
                  camp until morning
```

Sobre esse algoritmo, o enunciado do trabalho traz algumas premissas importantes:

- [P1] A estimativa da equipe de se conseguirão chegar até o próximo acampamento antes da noite sempre está correta;
- Considerando que a distância máxima que a equipe consegue viajar durante o dia antes do anoitecer é *d*, um conjunto de pontos de parada definidos pela equipe é válido se a distância entre cada par adjacente do conjunto, bem como entre a linha de largada e o primeiro ponto de parada e entre o último ponto de parada e a linha de chegada, é menor ou igual a *d*. [P2] O conjunto de pontos de parada definido pela equipe é válido.

---

## 3. Análise do Algoritmo:

Pelo enunciado do exercício, é desejado que seja demonstrado se esse algoritmo encontra o menor conjunto de pontos de parada que completa o *rally* ou não — ou seja, se o algoritmo é ótimo. Para isso, será analisada a propriedade de escolha *greedy* do algoritmo, que reside na estrutura de `if...else` do código em inglês estruturado apresentado na seção anterior — isto é, quando é decidido se é possível avançar para o próximo acampamento e alcançá-lo antes do anoitecer. Tomando a Premissa 1 (P1) da seção acima como

verdadeira, essa escolha local (míope) será, necessariamente, ótima. Logo, dada a estrutura do problema *greedy*, esse conjunto de escolhas locais ótimas fará com que a solução global também convirja para uma solução ótima. Poderia-se argumentar que existe a possibilidade de não haver como chegar de um acampamento a outro em um único dia, caso em que o algoritmo *greedy* não conseguiria convergir para uma solução. Entretanto, isso tornaria o circuito do *rally* inválido, o que é descartado se tomarmos a Propriedade 2 (P2) como verdadeira.

---

## 4. Implementação em Java e Tempo de Execução:

A implementação do algoritmo pode ser encontrada no subdiretório `src` da pasta do trabalho. Para fazer uma análise do tempo de execução, o programa foi executado 10 vezes com uma lista de cinco pontos de parada e foi tomada a média e a mediana dos tempos, de modo a tentar tornar a análise dos tempos o mais precisa possível.

| N° da execução | Tempo auferido |
|---|---|
| 1ª | 38 ms |
| 2ª | 29 ms |
| 3ª | 25 ms |
| 4ª | 45 ms |
| 5ª | 29 ms |
| 6ª | 28 ms |
| 7ª | 29 ms |
| 8ª | 33 ms |
| 9ª | 38 ms |
| 10ª | 50 ms |
| **MÉDIA** | **34,4 ms** |
| **MEDIANA** | **31 ms** |

Portanto, podemos concluir que, para a entrada fornecida, o programa leva pouco mais de 30 ms para executar o algoritmo *greedy*. Esse tempo considera apenas a execução do algoritmo, desconsiderando o tempo utilizado para operações de I/O.