# Trabalho 1 - PxNP

MC458
Pedro Carvalho Cintra
247315

---

## 1. O problema

Ocorrerá uma excursão no Parque exótico Nacional dos Picos (PxNP) e alguns turistas almejam passar por todos os picos do parque. Entre cada par de picos existe exatamente uma tirolesa que os conectam unidirecionalmente, ou seja, todas as tirolesas só podem ser usadas em uma única direção.

Nesse contexto, podemos modelar essa situação como um problema de grafos. O grafo em questão é um grafo dirigido simples onde cada vértice é um pico e vai ser representado por um número de 1 a $N$, sendo $N$ o número de picos no PxNP. Para organizarmos a excursão precisamos encontrar um caminho que passe por todos esses vértices uma única vez.

## 2. A solução

A partir das informações fornecidas, devemos desenvolver um algoritmo que devolva um vetor cujo os elementos sejam inteiros que representam o número do pico. Alem disso, a ordem desse vetor deve ser de tal forma que, ao percorrer o vetor sequencialmente da posição inicial para a posição final, tenhamos um caminho válido para a excursão.

Visto isso, inicialmente, e com base no $N$ recebido, criamos um vetor $V = [1, 2, \ldots, N]$ (lembrando que cada número no vetor é um vértice do grafo) e agora devemos permutá-lo para obtermos o vetor desejado.

É possível perceber uma grande similaridade com problemas de ordenação nesse caso, já que, de certa forma, devemos "ordenar" o vetor $V$ de acordo com alguma propriedade. Nesse caso, ao invés de ordenamos os números pela característica de serem maiores ou menores, ordenamos pelo fato de terem uma aresta ou não para o número seguinte. Porém, algumas propriedades não são satisfeitas como, por exemplo, em um problema de ordenação comum, se retirarmos um elemento em qualquer posição $i$ no meio do vetor ($1 < i < N$) ele continua sendo uma solução do problema. No caso do parque, o caminho é quebrado já que o vértice anterior a ele no vetor $V[i - 1]$ não necessariamente tem uma aresta conectando-o ao vértice posterior $V[i + 1]$ (lembrando que a arestas são direcionadas, por isso, para que haja conexão, a aresta deve ser no sentido $V[i - 1]$ até $V[i + 1]$).

### 2.1. O Projeto por Indução do Algoritmo

O algoritmo a ser desenvolvido deve receber o vetor $V$ criado e retornar a rota da excursão. Descrição do processo de indução usado para projetar o algoritmo:

- *Base*:

  Vetor com tamanho um. Trivial, só existe um único pico no parque.
- *Hipótese de Indução (h.i)*:

  Dado um vetor de tamanho $k$, onde $1 < k < n$, conseguimos organizá-lo de forma que cada elemento da posição inicial até a penúltima posição tenha uma aresta que o liga a posição da frente (formando um caminho), ou seja, cada vértice $V[i]$ tem uma aresta que o liga ao vértice $V[i + 1]$.
- *Passo Indutivo*:
  a. Seja $V$ um vetor de n elementos dividimos o vetor em duas partes de tamanho $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$. Por *h.i* conseguimos organizar esses dois vetores da maneira desejada. Precisamos agora combinar os dois vetores para obter a solução para o vetor de n elementos.

  b. O que faremos é montar o vetor final da seguinte forma: vamos percorrer os dois vetores ao mesmo tempo, olhando sempre para o primeiro elemento de cada um, conferir se o primeiro elemento do primeiro vetor tem uma aresta para o primeiro elemento do segundo vetor. Caso seja verdade, adicionamos o elemento do primeiro vetor no vetor final, caso contrário, adicionamos o elemento do segundo vetor no vetor final (é possível fazer essa combinação pois, no grafo em questão, caso não haja uma aresta de um elemento para o outro, necessariamente, há de existir uma aresta na direção contrária, visto que todos os pares são conectados em uma única direção). Após isso, o vetor do qual foi escolhido o elemento passa agora a ter como primeiro elemento o elemento seguinte e, novamente, fazemos a mesma verificação. Assim é feito sucessivamente até que um dos vetores chegue ao fim. Quando isso ocorrer, basta que armazenemos o restante dos elementos do outro vetor no vetor final, exatamente na ordem que eles se encontram (respaldo na *h.i*).

## 2.2. O Algoritmo

A partir do Projeto por Indução feito, conseguimos um pilar sólido para montar o algoritmo e, seguindo essa orientação, já temos também sua prova de corretude. O algoritmo é composto por duas funções: ‘excursionRoute’ e ‘combineRoutes’, que serão descritas a seguir.

O algoritmo ‘excursionRoute’ tem como entrada um vetor $V$ com os vértices do grafo, inteiros $e$ e $d$ que são, respectivamente, a posição inicial e final do intervalo do vetor que está sendo tratado. Essa função é responsável por alterar $V$ permutando-o das posições $e$ até $d$ de maneira a possibilitar a rota da excursão nesse intervalo. Para fazer isso ele realiza o processo descrito na primeira parte do *Passo Indutivo*, através dos seguintes passos: verifica se o intervalo $V[e..d]$ tem apenas um elemento (se $e = d$); caso seja verdade a função não realiza nenhuma instrução e volta para a chamada anterior; caso contrário, calculamos a posição do meio do vetor e acionamos a função novamente, para a primeira metade do vetor e depois para a segunda metade do vetor. Após ter passado por todas essas chamadas, acionamos a rotina ‘combineRoutes’ que combina essas duas partes desse intervalo do vetor.

Já a rotina ‘combineRoutes’ diz respeito a segunda parte do *Passo Indutivo* e tem como entrada um vetor $V$ com duas partes já permutadas, separadamente; além de inteiros $e$, $q$ e $d$ que são, respectivamente, a posição de início, meio e fim do intervalo do vetor que está sendo tratado. O algoritmo da função é responsável por combinar as partes do vetor $V$ em um único caminho válido. O vetor recebido é restrito ao intervalo $V[e...d]$, sendo dividido na parte $V[e... q]$ e $V[q + 1...d]$. Essas partes são soluções para o problema que contém o subconjunto de vértices presentes em cada partição. Devemos, então, encontrar a solução que satisfaz o conjunto que contém esses dois subconjuntos. Para isso criamos um vetor auxiliar $Aux$, armazenamos os elementos do intervalo $V$[e…d] nesse novo vetor e, após isso, vamos percorrer $V$, nesse mesmo intervalo, ao mesmo tempo que percorremos o vetor $Aux$ com dois índices que inicialmente referenciam os primeiros elementos da primeira e segunda parte de $Aux$ (índice $j$ e $i$ , inicialmente com valores $e$ e $q + 1$). Para cada iteração verificamos as seguintes situações: caso o segunda parte de $Aux$ já tenha sido inteiramente percorrida ($i > d$), então colocamos o elemento da primeira parte em $V[k]$ e auto incrementamos o índice dessa parte; caso isso não tenha acontecido, verificamos se existe uma aresta do elemento da primeira parte para o da segunda parte e se a primeira parte ainda não foi inteiramente percorrida ($e \leq q$), caso essas duas condições sejam satisfeitas, colocamos o elemento da primeira parte em $V[k]$ e auto incrementamos o índice dessa parte; caso contrário, colocamos o elemento da segunda parte em $V[k]$ e auto incrementamos o índice dessa parte.

## 3. A complexidade

Uma das principais características que devemos considerar quando procuramos desenvolver um algoritmo que seja eficiente é seu gasto de tempo para entregar uma solução. Por isso, vamos analisar a complexidade de tempo do algoritmo construído.

Verificando os passos do algoritmo feito vemos que, a função ‘excursionRoute’ realiza operações de tempo constante como atribuições e verificações, além de duas chamadas recursivas para intervalos do vetor que possuem metade do tamanho do intervalo do vetor original. Além disso, temos a rotina 'combineRoutes’ a qual realiza iterações de tamanho igual ao tamanho do intervalo fornecido na chamada atual, realizando diversas operações de tempo constante nessas iterações.

Dessa maneira, seja $T(n)$ o tempo de execução máximo do algoritmo entre todas as instâncias de tamanho $n = d - e + 1$, temos que:

$$T(n) = 1 \quad \text{, se } n = 1$$
$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad \text{, se } n \geq 2$$

Podemos encontrar uma solução para essa recorrência de várias maneiras. Por ter grande similaridade com recorrências que já conhecemos sua solução, vamos usar como palpite que $T(n) \in O(nlog(n))$, e substituir essa possibilidade de solução na recorrência.

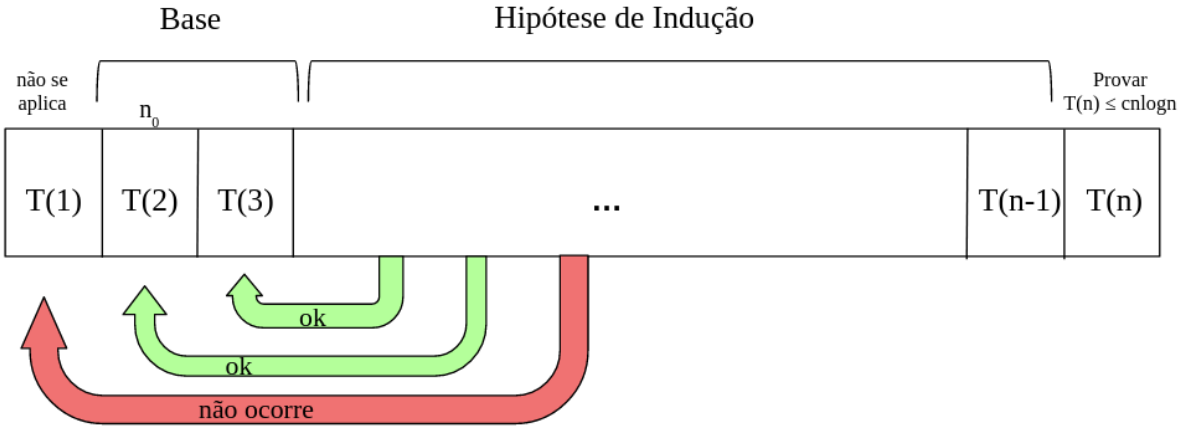
Para que o palpite seja de fato uma solução temos que provar que existe uma constante $c$ e um valor $n_0$ tal que: $T(n) \leq cnlog(n)$ para $n \geq n_0$ .

Vamo provar essa propriedade por Indução, ou seja, assumimos que a propriedade já vale para valores menores que $n$, nesse caso, $\lceil n/2 \rceil$ e $\lfloor n/2 \rfloor$:

$$
\begin{aligned}
&T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + an + b \leq \\
&\leq c\lceil n/2 \rceil log(\lceil n/2 \rceil) + c\lfloor n/2 \rfloor log(\lfloor n/2 \rfloor) + an + b \leq \\
&\leq c\lceil n/2 \rceil log(n) + c\lfloor n/2 \rfloor log(n/2) + an + b = \\
&= c\lceil n/2 \rceil log(n) + c\lfloor n/2 \rfloor (log(n) - 1) + an + b = \\
&= c(\lceil n/2 \rceil + \lfloor n/2 \rfloor) log(n) - cn/2 + an + b = \\
&= cnlog(n) - cn/2 + an + b \leq \\
&\leq cnlog(n) - cn/2 + an + bn \leq \\
&\leq cnlog(n)
\end{aligned}
$$

Dessa forma está provado que, para $c \geq 2(a + b)$, $T(n) \leq cnlog(n)$. Logo, para que $T(n) \in O(nlog(n))$ seja verdade, basta encontrarmos um $n_0$ que satisfaça essas condições.

Sabemos, ao analisar a recorrência, que todos os casos maiores recaem em T(2) ou $T(3)$ e que $T(n) \leq cnlog(n)$ não se aplica para $n = 1$. Porém, o que provamos se aplica para $n = 2$ e $n = 3$, a depender de valores das constantes $a$ e $b$, as quais não dizem respeito apenas a eficiência do algoritmo, mas, também, a arquitetura do sistema. Tomaremos, portanto, $n_0 = 2$ para que $T(n) \in O(nlog(n))$ esteja correto.




## 4. Conclusão

Assim sendo, conseguimos desenvolver um algoritmo eficiente para o problema de rota de excursão do PxNP, alem de provarmos sua corretude através do Projeto por Indução do algoritmo. Solucionar esse problema demandou uma análise detalhada de como iríamos modelar os picos do parque e também um pensamento muito próximo de uma solução por ordenação, dado que alocamos os picos em um vetor.