

# MC102QR - Algoritmos e Programação de Computadores

## Lab 12 - Algoritmos de Busca e Ordenação

**Prazo da atividade:** 4 de Julho de 2022

**Peso na nota:** 4 (9,76%)

O ano é 3022, a Terra agora se tornando uma civilização de nível 3 na escala de Kardashev é autorizada a fazer parte do acordo intergalático de comércio, permitindo então que qualquer entidade residente no planeta Terra realize acordos comerciais com qualquer uma das civilizações pertencentes a Liga Econômica Interplanetária (LEI).

O diretor executivo da *euComida*, um serviço terrestre de entregas de alimentos pela internet, decide que irá tomar proveito disso para expandir seus serviços de entrega para todas as galáxias pertencentes ao acordo, o que não será uma tarefa fácil. Dentre todos os avanços tecnológicos necessários para viabilizar a entrega de comidas por toda a galáxia, um ponto fundamental é desenvolver o sistema de logística intergalática.

Sendo as entregas agora feitas por robôs, já que eles podem ser programados para não dar “um grau” enquanto estão carregando o lanche de terceiros, não houve um grande problema em adaptá-los para viajar em espaço-motos. Entretanto, devido a fatores relativísticos, não faz sentido seus robôs entregadores levarem a encomenda em si para outras galáxias. Afinal uma pizza entregue com atraso de 100 mil anos não é tão apetitosa, mas sim levar portais de entrelaçamento quântico (que não sofrem dilatações do tempo) e distribuí-los em diversos pontos no tecido espaço-temporal, viajando através de buracos-de-minhoca (que sofrem dilatações do tempo), de forma que a qualquer momento sempre tenha um robô devidamente localizado para fazer essa entrega. Logo, a pizza que acabou de sair do forno na Terra seria enviada nesse portal e instantaneamente recriada do outro lado do universo.

Cada robô entregador recebe uma lista de rotas possíveis após fazer uma entrega, e decide então, qual a melhor rota que ele deve tomar dependendo das condições espaço-temporais do momento. O departamento de logística da *euComida* decidiu que para melhor cobrir o universo com entregadores robô, **a próxima melhor rota a ser tomada será percorrer a rota de maior distância possível**. Porém, alguns fatores devem ser levados em consideração para essa decisão:

- Diferente de motos tradicionais que usam gasolina como combustível, as espaço-motos usam antimatéria para realizar dobras-espaciais e poder viajar de um canto ao outro do universo em instantes, devido a natureza volátil desse composto, a capacidade máxima de um tanque de uma espaço-moto é de 50.0 hawkins, logo nem sempre ele poderá tomar a rota mais longa.
- Cada salto de dobra-espacial, consome todo o tanque de antimatéria, independente da distância viajada. Como estamos falando de viagens no espaço-tempo, 1 gota a mais de antimatéria no tanque pode deflacionar em proporções descomunais, fazendo com que a companhia perca milhões de bitcoins em questão de segundos. Logo, após cada viagem, a quantidade a

ser abastecida deve ser apenas o suficiente para realizar o trajeto que maximize a condição ótima descrita previamente.

## Tarefa

Sua tarefa consiste em programar qual rota um robô entregador deve fazer, de forma que faça a maior viagem possível seguindo a rota de entregas e dado uma quantidade limitada de antimatéria em seu tanque de combustível. A quantidade de antimatéria  $x$  que deve constar no tanque para viajar  $y$  anos luz é implicitamente dada por

$$\frac{\pi + ae^x - \zeta(bx + \pi)}{e^{-\sqrt{cx}} + d(2\pi^3 - x)} = y$$

onde  $a$ ,  $b$ ,  $c$  e  $d$  são parâmetros quânticos e relativísticos mensurados pela espaço-moto após cada salto e a função zeta é definida por:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

Como em 3022 ainda não descobrimos uma solução analítica para resolver a equação acima, a única abordagem até então é pela estimativa reversa, substituindo valores em  $x$  até achar um  $y$  desejado. Calcular a melhor rota a ser tomada por uma iteração simples (0 a 50 hawkins com uma precisão em micro-eletronvolt), para as possíveis distâncias em  $y$ , envolveria uma quantidade de operações tão colossal que o universo acabaria antes dessa estimativa ser concluída. Felizmente, devido ao fato da função ser crescente em  $x$  no intervalo de 0 a 50 hawkins, podemos usar a busca binária para encontrar um resultado ideal para o problema em um tempo viável.

**OBS:** Use os 100 primeiros termos do somatório na função zeta.

## Entrada

A primeira entrada consiste na quantidade de rotas  $N$  que o robô-entregador poderá escolher, em seguida é apresentado uma lista de `nome_do_planeta` seguido de `distancia_em_anos_luz`. As 4 linhas seguintes apresentam os parâmetros para o salto espaço-temporal. O programa deverá processar a lista de entrada até que  $N=0$ .

## Setup

Em Python, `1.1+2.2 == 3.3000000000000003` pois devido a representação finita de números, certos números não podem ser corretamente armazenados usando 64 bits<sup>1</sup>. Entretanto, como queremos estimar a precisão a nível de micro-eletronvolt (MeV)

---

<sup>1</sup> [Floating Point Arithmetic: Issues and Limitations — Python 3.10.4 documentation](#)

precisaremos de precisão até 36 casas após a vírgula. A biblioteca Decimal<sup>2</sup> nos permite abstrair a representação de números e fazer essa extensão da precisão, por isso use-a para calcular os valores esperados. Abaixo um exemplo de uso:

```
>> from decimal import getcontext, Decimal
>> getcontext().prec = 36
>> Decimal(3).exp()
Decimal('20.0855369231876677409285296545817179')
```

**OBS:** Para calcular os valores `ln`, `exp` e `sqrt`, use as funções padrões da biblioteca decimal, já para  $\pi$  use o arquivo dado `recipes_decimal.py` que se encontra junto aos arquivos de teste. Esse arquivo também disponível no Codepost, então ele não deve ser anexado no seu envio (e nem alterado).

Sugerimos você verificar se você implementou a função que dados `a`, `b`, `c`, `d` e `x`, calcula `y` corretamente verificando se você obtém os mesmo resultados abaixo no terminal (chamamos de `function` tal função):

```
>>> from lab12 import *
>>> a = b = c = d = Decimal(0.1)
>>> function(a, b, c, d, Decimal(50.))
Decimal('396343834401746822537.937852540166477')
>>> function(a, b, c, d, Decimal(25.))
Decimal('1842973461.22736342928049449664909884')
>>> function(a, b, c, d, Decimal(12.5))
Decimal('5084.29508581550533300332287259271110')
>>> function(a, b, c, d, Decimal(0.))
Decimal('0.286807630905896627280404747118816331')
```

## Saída

Para cada salto, o programa deverá imprimir, caso seja possível realizar um salto, o planeta que irá realizar o salto (o mais distante dentre os que for possível), seguido da quantidade de antimatéria que precisa abastecer para realizar esse salto com até 28 casas de precisão. e.g. `f"{tanque:.28f}"`. Caso não seja possível realizar o salto para nenhum dos planetas listados, o robô entregador está autorizado a dar um “grau” até a próxima entrega, nesse caso imprima “GRAU~~” na tela.

**OBS:** Sua impressão precisa ter precisão de 28 casas, logo realize a busca binária com erro máximo de  $10^{-32}$  para garantir que a aproximação esteja correta para 28 casas. O Decimal deve ter 36 casas de precisão para garantir que não haja problemas (`getcontext().prec = 36` - já definido no `recipes_decimal.py`).

**DICA1:** Para comparar dois valores decimais `a` e `b` para verificar se são "iguais", faça `abs(a - b) <= erro`, onde erro é o erro de precisão que você quer ter. Isso, de forma geral, não garante que eles são iguais até uma certa casa (por exemplo, 1.000000 e

---

<sup>2</sup> [Decimal fixed point and floating point arithmetic — Python 3.10.4 documentation](#)

0.999999 são bem próximos mas não são iguais em nenhuma casa), mas será suficiente nesse laboratório para garantir a resposta correta.

**DICA2:** Você não precisa fazer a busca binária para todos os planetas, raciocine sobre qual planeta você irá calcular a busca binária.

## Exemplos

### Exemplo 1:

#### Entrada

```
3
Arrakis
0.125820073422261803159696012183957511
Planeta Vegeta
1.059837123481723817238172387128373373
Betelgeuse V
0.073848848499192875475798123738548574
0.1
0.1
0.1
0.1
0
```

#### Saída

```
Planeta Vegeta
3.8534916525233666334702834889
```

### Exemplo 2:

#### Entrada

```
2
Battleworld
3747630.478267799739583301187520341700000000
Azeroth
3241506.114950603003232891687900394120000000
0.444425232145276183359783317428082228
0.608603590321059373380307988554704934
0.848647392059623140170288024819456041
0.959034793484243031080893615580862388
2
Coruscant
27842166.006375643717236124471569753300000000
The Known World
90772.810210751939247177825561933500000000
0.480221333571310804444465247797779739
0.320729314660765707500900134618859738
0.864187864934482075085497854161076248
0.132273685566122622958573629148304462
0
```

## Saída

Battleworld  
19.6524024105247523493652232567  
Coruscant  
19.6029661492300127448761460659

### Exemplo 3:

## Entrada

[illegible]

## Saída

```
migus pumbus
30.8844694323183345377417566074
collectables particles ieee
5.8592424354681998707783661898
GRAU~~
```

## Submissão

Você deverá submeter no CodePost, na tarefa Lab 12, um arquivo com o nome `lab12.py`, contendo todo o seu programa. Após o prazo estabelecido para a atividade, será aberta uma tarefa Lab 12 - Segunda Chance, com prazo de entrega até o fim do semestre.